ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO EN INFORMÁTICA

Título del proyecto:

SISTEMA COLABORATIVO DE CLASIFICACIÓN DE IMÁGENES MÉDICAS BASADO EN FOLKSONOMÍAS

Adrián Ciordia Galar

Tutores: José Javier Astrain Escola

Alberto Córdoba Izaguirre

Pamplona, 9 de Septiembre de 2011

Agradecimientos.

Quiero agradecer a mis tutores Joseja y Alberto su guía y apoyo a lo largo del desarrollo del proyecto. También quisiera agradecer a Patxi su ayuda, tanto a la hora de resolverme dudas sobre el trabajo de su tesis, como a la hora de sugerirme modificaciones para mejorar el rendimiento de la aplicación. Y, por último, agradecer a mis amigos Nacho e Iker por prestarse tardes enteras a calificar imágenes médicas en su tiempo libre, pudiendo estar haciendo cualquier otra actividad más divertida.

Índice:

Índice de Figuras:

1. Introducción.

1.1. Introducción

En un principio las *Webs* de *Internet* no eran más que páginas estáticas, normalmente programadas en *HTML*, que ofrecían al usuario cierta información, prácticamente sólo texto, que no era actualizada con asiduidad. Con el paso del tiempo estas páginas fueron evolucionando volviéndose dinámicas gracias a la utilización de otros lenguajes de programación como *PHP* o similares, ofreciendo contenido multimedia como sonidos o vídeos, interfaces más amigables para el usuario y, permitiendo a éste generar su propio contenido sin tener que ser un experto. Este cambio en la forma de crear y usar la *Web* es lo que se denominó *Web 2.0*. Esto condujo a la aparición de páginas Web que ofrecen funcionalidades radicalmente diferentes a las que originalmente se habían planteado que tuvieran. Con la *Web 2.0*. aparecen los blogs, recopilan textos de los usuarios ordenados cronológicamente, las Wikis, enciclopedias virtuales que pueden ser actualizadas por cualquier usuario, motores de búsqueda, permiten al usuario buscar páginas en función de una serie de palabras clave elegidas por el mismo, redes sociales, en las que el usuario establece vínculos con otros usuarios representando relaciones interpersonales de todo tipo creando grupos de personas con las que pueden comunicarse, páginas de *social bookmarking*, donde el usuario tiene la posibilidad de etiquetar una serie de recursos, y otros muchos tipos de páginas y aplicaciones *Web*. Ejemplos de dichas páginas son:

- *YouTube*[1]: ofrece al usuario la capacidad de almacenar, visualizar y compartir vídeos con otros usuarios.
- *Flickr*[2]: da la posibilidad al usuario de almacenar, visualizar, compartir y vender imágenes y fotografías a otros usuarios.
- *Facebook*[3]: es una red social.

[1] http://www.youtube.com/
[2] http://www.flickr.com/
[3] http://www.facebook.com/

- *Delicious*[1]: permite al usuario almacenar, compartir y etiquetar marcadores (sitios Web).
- *Google*[2]: es un motor de búsqueda.
- *Wikipedia*[3]: es una Wiki.
- *Blogger*[4]: es un sitio Web que permite la creación de blogs.

Como se ha dicho con anterioridad, en todo este tipo de páginas *Web* tiene mucha importancia el usuario como generador de nuevo contenido en *Internet* y debido a eso ha aumentado en gran medida la información almacenada en la Red.

Con el surgimiento de esta gran cantidad de datos aparecieron consigo los problemas de representar y clasificar dicha información. En la actualidad se está utilizando bastante el etiquetado social o colaborativo (*collaborative tagging*) sobre otros sistemas de clasificación de la información. Este sistema permite a los usuarios asociar una serie de etiquetas de texto a los recursos presentados en el sistema, de forma que pueden plasmar lo que dichos recursos les sugieren dotándolos de cierto contenido semántico. Esto, también, establece una relación directa entre recursos con etiquetas coincidentes de forma que se crean grupos de recursos.

A este conjunto de recursos, usuarios, etiquetas y anotaciones de etiquetas es lo que se denomina folksonomía. Hoy en día las folksonomías son el sistema de representación de información predominante en Internet. Se han impuesto sobre otros sistemas de representación de la información como taxonomías u ontologías. Esto es debido a que es un sistema sencillo, no jerárquico, de actualización fácil y continua, actualizable por un grupo amplio de personas sin necesidad de que tengan conocimientos avanzados.

Pero parte de lo que ha hecho popular el uso de las folksonomías también es parte de su debilidad como sistema de representación de la información. Debido a su forma de actualización, a su diseño interno y al uso de un vocabulario no controlado, es muy proclive a problemas de ambigüedad, polisemia y sinonimia. Es decir, que el sistema permite la anotación de etiquetas que pueden admitir distintos significados o etiquetas con una semántica idéntica pero diferente sintáctica.

En contraposición a las folksonomías existe otra forma de representación de conocimiento que se ha intentado implantar en la Web pero que debido a sus exigencias para su creación y mantenimiento no lo ha conseguido satisfactoriamente. Estas son las ontologías. Son una representación formal del conocimiento constituida por conceptos y relaciones en el contexto de un dominio. En comparación con las folksonomías, las ontologías son sistemas complejos con estructura jerárquica, tienen un coste de creación y actualización elevado que sólo pueden llevar a cabo un grupo reducido de expertos. Por otro lado, carecen prácticamente de ambigüedad, los problemas de sinonimia y polisemia son controlables y proporcionan la capacidad de separación del significante del significado.

El uso de ontologías es uno de los pilares fundamentales donde se asientan las bases de la *Web Semántica*. Ésta es considerada como una extensión de la *World Wide Web* en la que las propias máquinas son capaces de entender el significado del contenido de la Red. Esto

[1] http://www.delicious.com/
[2] http://www.google.com/
[3] http://en.wikipedia.org/
[4] http://www.blogger.com/

se llevaría a cabo mediante la inserción, en las páginas *Web* tradicionales, una serie de metadatos leíbles por máquinas que describieran la página en sí y su relación con las demás. Esto permitiría a una serie de programas y aplicaciones procesar la información de la Red con mayor facilidad y les posibilitaría realizar tareas, sobre éstas, de manera automática en nombre de los usuarios.

Así, por un lado tenemos la *Web 2.0.* o Web social donde hay una gran cantidad de información que está constantemente generándose y actualizándose, pero que debido a que está almacenada de una forma no estructurada y no controlada, carece de gran parte de la potencia de cálculo de nuevo conocimiento que podría tener. Y por otro lado está la *Web Semántica* con la información presentada de forma estructurada, formal y documentada, pero cuya generación y actualización es compleja, llevada a cabo sólo por expertos y realizada de manera puntual.

Muchos autores se han planteado el combinar o suplir el espacio entre estas dos formas de representación del conocimiento aunando así la potencia de las ontologías con la flexibilidad de creación y actualización de las folksonomías. En este aspecto ya se han presentado diferentes trabajos al respecto o intentando mejorar la navegabilidad de las folksonomías. Por ejemplo, en [1] Tom Gruber presenta lo que denomina Sistemas de Conocimiento Colectivo (*Collective Knowledge Systems*) en los cuales se pretende aprovechar la información generada por los usuarios para procesarla, estructurarla, obtener nueva información y mejorar con ello la experiencia del usuario. En [2] presentan un método con el que clasificar etiquetas en categorías de una ontología preestablecida y posteriormente explorar los recursos de acuerdo a las categorías asignadas a sus etiquetas.

1.2. Descripción del Problema.

1.2.1. Folksonomías.

Como ya se ha comentado con anterioridad, en la actualidad, los usuarios de páginas *Web* han pasado de ser meros observadores a formar parte activa en la creación y actualización de dichas páginas. Tras este cambio de dirección en el flujo de la información en la *Web*, ésta se pobló de gran cantidad de datos generados por los propios usuarios, que debido a su gran magnitud era relativamente difícil acceder a ellos de una forma eficaz y sencilla. A consecuencia de esto era necesario encontrar un método eficaz para clasificar y representar toda esa cantidad de información. Se acabaron imponiendo las folksonomías y el etiquetado colaborativo que iba muy parejo a esta nueva filosofía de entender *Internet*

Las folksonomías son un sistema de clasificación y de representación del conocimiento que se basa en la asociación de etiquetas de texto, por parte de los usuarios, a recursos pertenecientes al propio sistema, que pueden haber sido añadido por los mismos usuarios o por otros. El realizar dichas anotaciones de etiquetas permite establecer vínculos o relaciones entre diferentes recursos.

Un ejemplo sencillo de esto sería el sitio *Web Flickr*[1]. En esta página un usuario registrado podría subir una fotografía, imaginemos, de un coche. Al introducir la fotografía en el sistema, al usuario se le permitiría introducir una serie de etiquetas asociadas a la imagen. En este hipotético caso nuestro usuario va a introducir la etiqueta "coche" con la que la imagen quedaría relacionada con otras que también estuviesen anotadas con dicha etiqueta. Entonces, ahora podríamos buscar aquellas imágenes etiquetadas con "coche" y obtendríamos entre otras nuestra imagen insertada con anterioridad.

Otros ejemplos donde se usa actualmente folksonomías para la representación del conocimiento colectivo son *Delicious*[2], una página de Internet donde los usuarios pueden compartir sus marcadores de páginas Web y clasificarlos mediante etiquetas, o *Last.fm*[3], una radio vía Internet donde la gente etiqueta canciones y se usa esta información para ofrecer música acorde a los gustos del oyente.

La mayor ventaja que ofrecen las folksonomías es la posibilidad que brindan al usuario de utilizar cualquier etiqueta de texto para anotar cualquier recurso del sistema, es decir, que las etiquetas no pertenecen a un vocabulario controlado. Por tanto, se le deja al usuario una libertad total para clasificar los recursos como le plazca. Por otro lado, esto mismo supone uno de los puntos flacos, ya que implica problemas como ambigüedad, qué no esté claro el significado de la etiqueta, polisemia, que la etiqueta pueda contener múltiples significados, y sinonimia, que etiquetas con una sintáctica diferente tengan un significado semántico idéntico.

Otra de las ventajas que ofrecen las folksonomías es el hecho de que la creación y evolución de las mismas se realiza por un grupo amplio de personas de forma continuada. Esto permite que el coste de creación para los autores de un sistema que las utilizan sea mínimo, además de que se consigue una representación popular del área del conocimiento tratado, en el sentido de que se toman en consideración los puntos de vista de todos los usuarios. De esta misma forma las folksonomías suelen estar actualizadas con los últimos conceptos de los temas que recogen ya que son los propios usuarios de las mismas las que las van actualizando.

A parte de las ventajas vistas anteriormente, las folksonomías, también presentan una serie de desventajas. La más representativa es la falta de estructura interna para organizar y clasificar las etiquetas de la folksonomía, que si bien es una ventaja a la hora de facilitar la inserción de nuevos datos al usuario, es un inconveniente a la hora de procesar el contenido semántico de la misma. Por ejemplo podríamos tener una imagen con la etiqueta asociada "gato" con la que una persona sería capaz de distinguir si se refiere al animal o a la herramienta solamente con mirar la imagen, en cambio con la información almacenada en una folksonomía, un ordenador, consideraría iguales a las imágenes de gatos como animal que las imágenes de gatos como herramienta. En cambio si la estructura fuese más compleja podría haber dos identificadores "g1" y "g2" para referirse al animal y a la herramienta respectivamente, ambos dos asociados a un valor de etiqueta "gato". Cada uno de estos identificadores pertenecerían a dos grupos más amplios, "g1" pertenecería a "Animal" y "g2" a "Herramienta". Entonces podríamos anotar las imágenes con la etiqueta "gato" usando los identificadores "g1" y "g2" con lo que, ahora sí, una máquina sería

[1] http://www.flickr.com/
[2] http://www.delicious.com/
[3] http://www.last.fm/

capaz de separar las imágenes de animales de las de herramientas referentes a la sintáctica "gato".

Las folksonomías pueden clasificarse principalmente en dos tipos: folksonomías *broad* y folksonomías *narrow*. En las folksonomías de tipo *broad* se permite a cualquier usuario anotar cualquier etiqueta en cualquier recurso. Por otro lado, las folksonomías de tipo *narrow* sólo permiten etiquetar un recurso al autor de su publicación o a un grupo reducido de usuarios.

Como puede entenderse, debido a esa diferencia de etiquetado va a haber una diferencia de calidad en la semántica asociada a los recursos. En las folksonomías de tipo *broad* va a haber un gran número de usuarios que van a ir etiquetando cada recurso. Conforme se vayan añadiendo etiquetas de diferentes usuarios a un recurso va a haber algunas de ellas que se van ir repitiendo y otras no. Aquellas que más se repitan van a ser las que van a dar el valor semántico al recurso ya que son las más usuarios han usado para describir el mismo. En cambio, en las folksonomías de tipo *narrow* al estar el etiquetado de un recurso limitado a una o unas pocas personas el contenido semántico del mismo se va a ver reducido a lo que les evoque a dichas personas el recurso. Por tanto, existe el riesgo de ser menos objetivo que en una folksonomía de tipo *broad*.

En resumen, las folksonomías ofrecen una manera fácil e inmediata de clasificar los recursos de un sistema debido a su sencillez y falta de una estructura compleja interna. Pero debido a esto mismo se vuelve más difícil el procesamiento de la información de la misma y se desaprovecha gran parte de su semántica.

1.2.2. *Web Semántica* y Ontologías.

Tim Berners-Lee presentó en 2001 una nueva forma de entender la Red, la *Web Semántica* [3]. En este artículo se impulsaba el desarrollo de una nueva *Web* como una extensión de la ya existente. Actualmente la *Web* está diseñada de forma que sea entendible por las personas, pero este nuevo concepto lo que pretende es que también lo sea por máquinas y ordenadores. El objetivo de esto es conseguir que, tanto personas como ordenadores, trabajen cooperativamente de una forma mejor y más eficiente.

Para que los ordenadores sean capaces de entender el contenido de la *Web* es necesario que éstos tengan acceso a colecciones de información estructurada que representen la parte del conocimiento que van a tratar, y una serie de reglas de inferencia que les permitan llevar a cabo razonamientos de forma automatizada. El objetivo de esta iniciativa no es que los ordenadores puedan entender el propio documento publicado en la *Web* sino que sean capaces de comprender una serie datos semánticos asociados a los mismos. Tim Berners-Lee propone en [3] el uso de tecnología *XML* y *RDF* para describir las propiedades de los objetos. *XML* dotaría de estructura a los documentos semánticos y *RDF* se encargaría de describir el significado.

Una vez que ya somos capaces de definir un contenido semántico en los documentos surge el problema de unificar los términos que se van a usar a la hora de describir la semántica de los mismos. Es decir, que las páginas *Web* y documentos usen los mismos términos para expresar los conceptos, o que al menos sean capaces de establecer equivalencias entre los

vocabularios usados en uno y otro lado. Para resolver este problema entran en juego las ontologías.

Una ontología es una representación formal del conocimiento, dentro de un dominio, que define una serie de términos y las relaciones entre ellos. En ellas pueden definirse clases, con una serie de propiedades que pueden ser heredadas por las subclases que desciendan de las mismas. Así mismo, en las otologías también pueden expresarse reglas de inferencia de tipo *if – then* que es lo que va a permitir a los programas llegar a hacer deducciones en base a la información dada.

Actualmente, la principal forma de establecer relaciones entre los elementos que conforman la Red se constituyen mediante el uso de enlaces, pero gracias a esta forma de representación del conocimiento asociada a los documentos y páginas *Web* se establecería un nuevo tipo de relaciones que no sólo indicaría que elemento está relacionado con qué otro sino que también la naturaleza de dicha relación.

Tim Berners-Lee, en el artículo mencionado con anterioridad, también describe los que él llama "agentes" dentro del concepto de la *Web Semántica*, que serán los que, según él, muestren el verdadero potencial de la *Web Semántica*. Estos "agentes" son descritos como programas o aplicaciones, desarrolladas por diferentes personas, que recolectan información de la *Web*, la procesan y comparten con otros programas, haciendo uso de los datos semánticos. Dichos agentes deberían estar preparados para trabajar de forma colaborativa unos con otros, de forma que la ejecución de un proceso implicaría que un agente hiciese una serie de consultas a otros agentes, y estos a su vez las hicieran a otros y así sucesivamente, delegando a cada uno de los agentes la construcción de parte de información requerida por parte del usuario final.

Como se ha dicho con anterioridad, las ontologías son una parte fundamental de la *Web Semántica* ya que son las que dotan de cierta universalidad de los términos utilizados, los clasifica y los relaciona entre sí. Pero en contraposición de esta potencia semántica que las caracteriza también consta de una serie de deficiencias que dificultan su uso. Uno de los principales problemas que presentan las ontologías es que su creación es llevada a cabo por un grupo reducido de expertos en el dominio en que se va a centrar la información recogida. Esto hace que su creación y posterior actualización sea muy lenta y costosa. Por tanto, esto también hace que sea poco adaptativa a los diferentes cambios que se producen en el área de conocimiento que trata, haciendo que quede desfasada u obsoleta.

1.2.3. Imágenes médicas.

Como va a poder verse en apartados posteriores, las imágenes médicas van a ser un elemento de importancia para el desarrollo del proyecto. Por tanto, en esta sección, van a explicarse de forma escueta en qué consisten dichas imágenes.

Según la *Wikipedia*, se llama imagen médica al conjunto de técnicas y procesos usados para crear imágenes del cuerpo humano, o partes de él, con propósitos clínicos (procedimientos médicos que buscan revelar, diagnosticar o examinar enfermedades) o para la ciencia médica (incluyendo el estudio de la anatomía normal y función).[1]

[1] http://es.wikipedia.org/wiki/Imagen_médica

Hay muchas técnicas a la hora de obtener dichas imágenes. A continuación se van a explicar algunas de ellas.

Las radiografías son proyecciones en dos dimensiones hechas con rayos X u otras componentes de alta frecuencia del espectro electromagnético, que permiten observar los huesos y algunos órganos del cuerpo.

Figura 1.1. Radiografía.

Una especialización de las radiografías son las fluoroscopias. Al paciente se le introduce una sustancia que hace de contraste y permite ver órganos o partes de ellos que sin esa sustancia aparecerían como sombras en la radiografía.

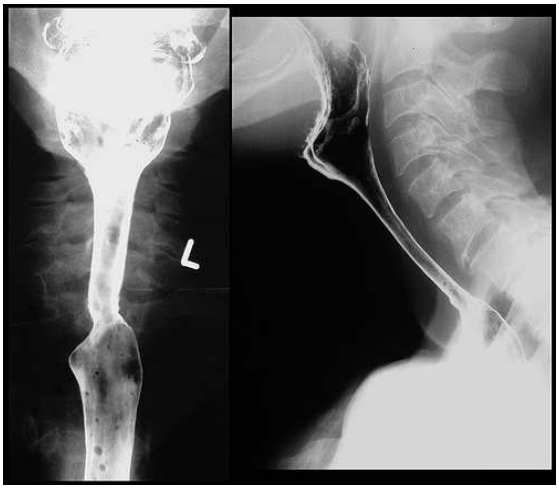


Figura 1.2. Fluoroscopia.

Otra de las técnicas utilizadas es el TAC (Tomografía Axial Computerizada). El TAC se basa en construir imágenes tridimensionales, mediante cálculo computacional, a partir de un número considerable de radiografías en dos dimensiones.

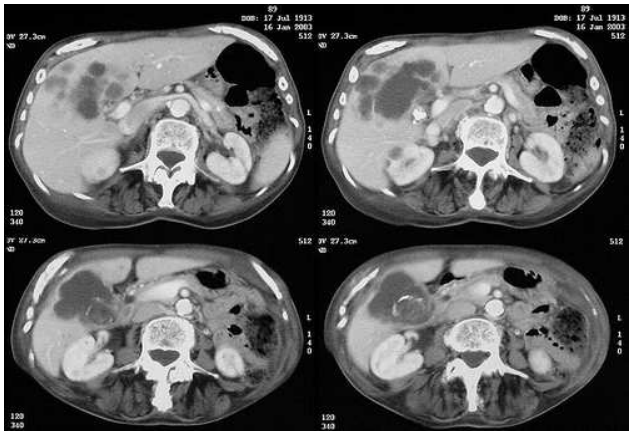


Figura 1.3. TAC.

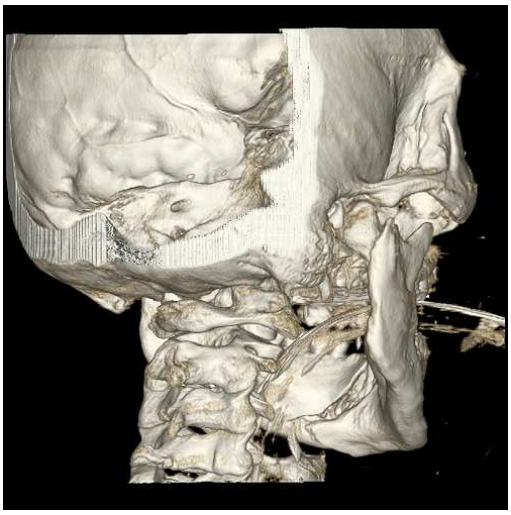


Figura 1.4. TAC.

Otras de las formas de poder visualizar la estructura interna de los cuerpos es el uso de ecografías. Las ecografías son un procedimiento que usa ultrasonidos para construir una imagen, en vez de los rayos X que usan las técnicas anteriormente vistas. Un elemento, llamado transductor, emite esos ultrasonidos. Éstos atraviesan el cuerpo y son parcialmente reflejados ante cambios de densidad del mismo, como puede ser el cambio de un tipo de tejido a otro. El eco producido es recibido por el transductor, haciendo que este vibre, y dicha vibración es transformada en una señal eléctrica que a su vez es enviada a un ordenador. Esa máquina transforma los pulsos eléctricos en una imagen digital que muestra el interior del cuerpo.

Figura 1.5. Ecografía.

Una forma más de ver el interior de un cuerpo son las resonancias magnéticas. Estas estimulan los átomos de hidrogeno de las moléculas de agua del cuerpo mediante el uso de campos magnéticos. Los cambios producidos en dichos átomos son registrados por el escáner y, a su vez, utilizados para generar una imagen.

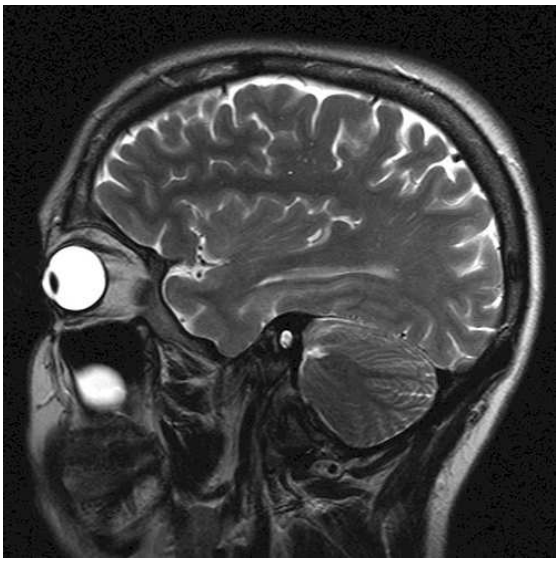


Figura 1.6. Resonancia Magnética

Por último, la medicina nuclear hace uso de radiofármacos que contienen isótopos radiactivos. Dichas sustancias son introducidas en el organismo y son capaces de localizar órganos o tejidos específicos. La interacción de los radiofármacos con el cuerpo es registrada por un detector de radiación y dicha información es almacenada digitalmente. Posteriormente, se procesan los datos y con ellos se construyen imágenes del organismo. Los resultados obtenidos permiten estudiar el funcionamiento de los órganos y mostrar su comportamiento molecular.

Una de las técnicas más concretas que hace uso de radiofármacos son las tomografías por emisión de positrones o PET. Éstas hacen uso de la radiación gamma, producida por la

interacción del radiofármaco con el organismo, para construir una tomografía, en vez de los rayos X del TAC.

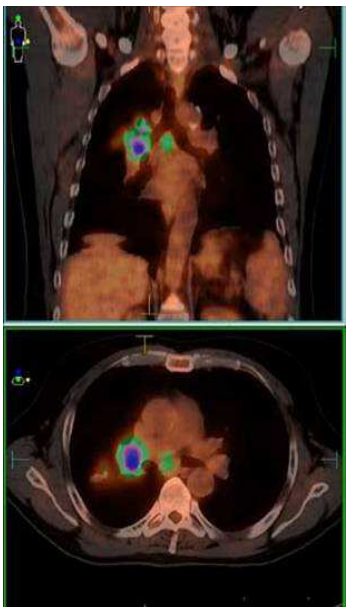


Figura 1.7. PET

Estos son algunos de los tipos de imágenes médicas con los que va a tratar la aplicación desarrollada en este proyecto. Se han explicado sólo aquellas que se han considerado más representativas debido a que no se considera como un prerrequisito ni un objetivo del proyecto la comprensión de dichas imágenes.

1.3. Objetivos Perseguidos.

El objetivo principal de este proyecto es conseguir clasificar de manera automática los recursos de una folksonomía de imágenes médicas en una serie de categorías. Con ello se quiere mejorar la representación de la información y obtener nuevas formas de consultarla, sin ser alterado el método con que los usuarios anotan los recursos. De esta manera se conseguiría mantener las ventajas que tiene la creación de folksonomías, pero se mejoraría la estructura en la que están organizados los datos y la forma es que están relacionados unos con otros. Para ello se ha hecho uso de parte del trabajo presentado en la tesis doctoral de Francisco Echarte [4].

Como objetivos adicionales se quiere realizar la implementación de parte del programa presentado en dicha tesis de forma que sea de fácil portabilidad y multiplataforma. Además se quiere aprovechar la modularidad de dicho programa para facilitar que, en un futuro, puedan cambiarse los métodos, fórmulas o algoritmos que se encargan de realizar los cálculos en cada uno de los componentes de la aplicación.

1.4. Estado del Arte.

Como ya se ha comentado en la introducción de esta memoria, ya ha habido intentos, por diversos investigadores, de acortar el espacio que separa las folksonomías de las

ontologías. Tom Gruber nos presentaba sus "Sistemas de Conocimiento Colectivo" [1] que pretenden aprovechar la información que los usuarios generan en *Webs* sociales para procesarla y generar con ella nuevo conocimiento que a su vez vuelto a poner a disposición de los usuarios. Por otro lado, Rabeeh Abbasi propone *T-ORG* [2], un método con el que es posible clasificar automáticamente un conjunto de etiquetas en una serie de términos pertenecientes a una ontología dada, previamente refinada. Este método hace uso de patrones lingüísticos para construir consultas con las etiquetas. Con ellas obtiene de *Google* los "abstract" de los resultados de búsqueda, que luego compara con el contexto de la etiqueta y, si supera un umbral, extrae la categoría del *abstact* clasificando en ella la etiqueta. Tras esta clasificación es posible navegar entre los recursos que tienen dichas etiquetas usando los términos en los que han sido clasificados. Miao Chen presenta en [5] una metodología que permite extraer relaciones entre pares etiquetas de una folksonomía. Estas se extraen de los textos de los resultados de un motor de búsqueda sobre las consultas hechas del conjunto de parejas de etiquetas que más aparecen conjuntamente en recursos de la folksonomía. Hace uso de *NLP* (*Natural Language Processing*) para analizar los textos y algoritmos de aprendizaje automático para clasificar las relaciones. Sofia Angeletou muestra en [6] *FLOR*, un algoritmo de enriquecimiento de folksonomías. Este método construye una estructura semántica a partir de un conjunto de etiquetas, haciendo uso de consultas a *WordNet*[1] y a ontologías *online*. Silvia Bindelli propone en [7] un sistema de búsqueda sobre un conjunto folksonomías que hace uso de ontologías para refinar o concretar la definición de las consultas. Patrick Schmitz en [8] hace uso de modelos estadísticos para crear árboles de términos jerarquizando etiquetas de un conjunto de imágenes extraídas de *Flickr*[2]. Francisco Echarte presenta en [4] *ACoAR*, un método para la clasificación automática de recursos de una folksonomía. Para ello hace uso de técnicas de *clustering* como *k-means* y medidas de similaridad como el coseno, entre otras. Además, el propio diseño modular del sistema permite el fácil intercambio de componentes del mismo. Gracias a esto pueden cambiarse de forma sencillas los métodos o algoritmos que se usan para calcular cada parte del proceso.

Estos son sólo algunos ejemplos del trabajo que se lleva haciendo en la comunidad científica en la línea de enriquecer la semántica de las folksonomías o de transformar éstas, lo mejor posible, en ontologías.

Por otro lado, ya se han ido elaborando otologías sobre términos médicos. Un ejemplo de esto es *Snomed-CT* [3](*Systematized Nomenclature of Medicine Clinical Terms*) que es considerada una de las más importantes y completas terminologías clínicas multilingüe. Combina desde términos de ciencias básicas y bioquímica hasta especialidades médicas y contenidos de atención primaria. Este producto nace de la unión de *Snomed RT*, una terminología médica, desarrollada por el *College of American Pathologists* (*CAP*), y *CTV3* (*Clinical Terms Version 3*), desarrollada por la *National Health Service* (*NHS*) del Reino Unido. Actualmente *Snomed-CT* está siendo mantenida y distribuida por la *IHTSDO* (*International Health Termilogy Standards Development Organisation*), una organización encargada de administrar diversos estándares relacionado con las ciencias de la salud. *Snomed-CT* está siendo impulsada por los gobiernos de diferentes países como Australia, Canadá, Estados Unidos, España, Nueva Zelanda, etc., para convertirse en el estándar de referencia dentro del dominio de las terminologías médicas.

[1] http://wordnet.princeton.edu/
[2] http://www.flickr.com/
[3] http://www.ihtsdo.org/

1.5. Solución Propuesta.

La solución propuesta consiste en realizar una implementación de *ACoAR* capaz de clasificar los recursos de una folksonomía de imágenes médicas en un conjunto de conceptos.

2. Análisis y Diseño.

Como ya se ha comentado en la introducción de esta memoria hay distintos objetivos que se han perseguido durante el desarrollo de este proyecto. El objetivo principal sería el conseguir clasificar una serie de imágenes médicas en una serie de conjuntos. Para ello se va hacer uso del trabajo realizado en [4] implementando parte del sistema que han desarrollado. Partiendo de esto, vemos que dicho sistema necesita como datos de entrada una folksonomía, por tanto, lo primero que hemos tenido que hacer ha sido el obtener una folksonomía de imágenes médicas.

A continuación se detalla el análisis y el diseño de las aplicaciones desarrolladas en el proyecto.

2.1. Análisis y Diseño de *FlickrBackup*.

El primer problema que se nos planteó durante el desarrollo del proyecto fue precisamente la obtención de los datos de entrada de la aplicación que queríamos desarrollar, la folksonomía de imágenes médicas. Cómo no nos ha sido posible que ninguna institución médica nos proporcionase dichos datos, los hemos tenido que obtener por otros medios. Para ello se decidió hacer uso de *Flickr*[1], una red social en que la gente comparte y etiqueta imágenes. Vimos que en esta página había cierta cantidad de imágenes médicas que podrían sernos de utilidad. Además, la información ya estaba presentada en forma de folksonomía. Por tanto, el siguiente paso era el descargarnos de dicha página las imágenes médicas con toda la información que tuviesen asociadas, como etiquetas, etc., y así crear en una máquina un subconjunto de la folksonomía de *Flickr* sólo del dominio que nos interesa a nosotros, las imágenes médicas. Buscado la forma más eficiente de hacer esto vimos que *Flickr* tiene implementado un *API*[2] para desarrollar aplicaciones que hagan uso de la información contenida en su sistema. A su vez vimos que dicha *API* estaba adaptada

[1] http://www.flickr.com/
[2] http://www.flickr.com/services/api/

a java en una librería de clases llamada *flickrj*[1]. Por tanto, decidimos construir una aplicación que automatizase el proceso descarga de la folksomía haciendo uso de dicha librería.

2.1.1. Análisis de Requisitos.

A continuación se van a enumerar los distintos requisitos funcionales y no funcionales de la aplicación de descarga y almacenamiento de imágenes de *Flickr*.

2.1.1.1. Requisitos Funcionales.

- Tiene que descargar y almacenar imágenes de *Flickr* con toda su información asociada: usuario que la subió, etiquetas anotadas y notas.
- Debe permitir al usuario introducir qué etiquetas deben tener las imágenes a descargar y poder establecer un número máximo de imágenes a descargar por ejecución.

2.1.1.2. Requisitos No funcionales.

- Tiene que poder ejecutarse en *Windows XP* y, a ser posible, en otros sistemas operativos.
- Tiene que ofrecer conectividad a *MySQL*.
- Debe poder comunicarse con el *API* de *Flickr*.
- Tiene que almacenar las imágenes, en distintos tamaños, en el disco duro del ordenador.
- Debe guardar la información asociada de cada imagen en una base de datos.

2.1.2. Casos de Uso.

En esta sección se muestran y describen los diagramas de casos de uso de la aplicación en sus diferentes niveles de iteración.

2.1.2.1. Iteración *Draft*.

El usuario interactúa con la aplicación *FlickrBackup*.

[1] http://flickrj.sourceforge.net/

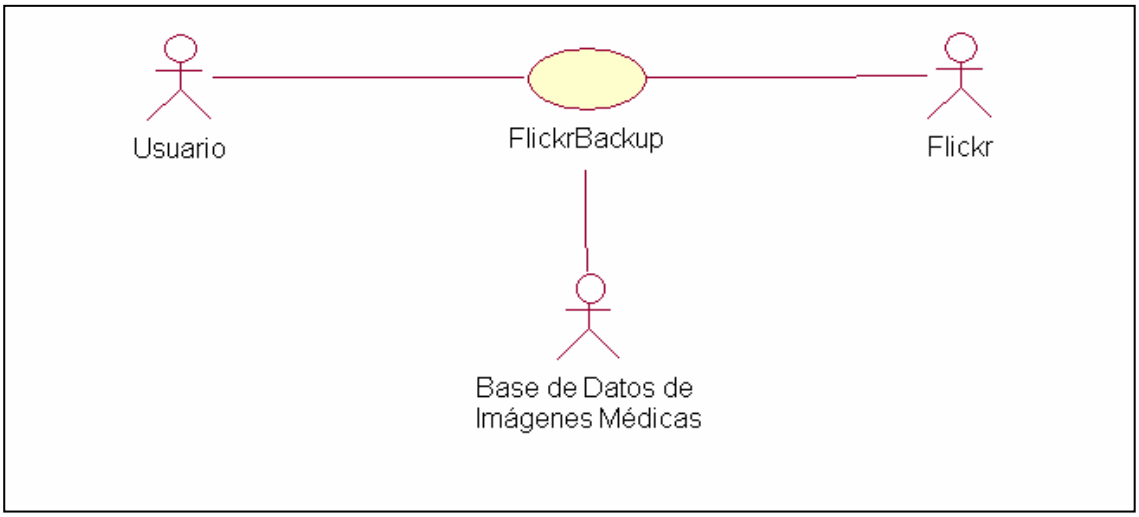


Figura 2.1. Iteración Draft de FlickrBackup. Diagrama de casos de uso.

Actores Principales:

El actor principal es el usuario de la aplicación.

Actores Secundarios:

Los actores secundarios son el sistema de *Flickr* y el sistema gestor de bases de datos que contiene la base de datos de imágenes médicas.

Personal Involucrado e Intereses:

Usuario: desea descargar imágenes de *Flickr*.

Precondiciones:

El usuario debe ejecutar la aplicación y tener una serie de palabras clave a buscar, y la aplicación debe tener conexión con la base de datos.

Poscondiciones:

Se habrán almacenado en disco las imágenes y su información asociada en la base de datos de aquellos recursos de *Fickr* que contengan las etiquetas introducidas como entrada.

Flujo Básico:

1. El usuario inicia la aplicación.
2. Introduce las etiquetas por las que se van a buscar recursos en *Flickr*.
3. Introduce un número máximo de imágenes a descargar, o introduce 0 para no definir un tope.

4. La aplicación busca los identificadores de las imágenes de *Flickr* que cumplen con los requisitos.
5. Se va almacenando cada una de las imágenes y su información asociada.
6. Se cierra la aplicación.

Flujo Alternativo:

No hay flujo alternativo.

2.1.2.2. Iteración 1: *FlickrBackup.*

El usuario introduce una serie de parámetros de entrada y la aplicación descarga y almacena las imágenes de *Flickr* que cumplen los requisitos definidos por dichos parámetros.

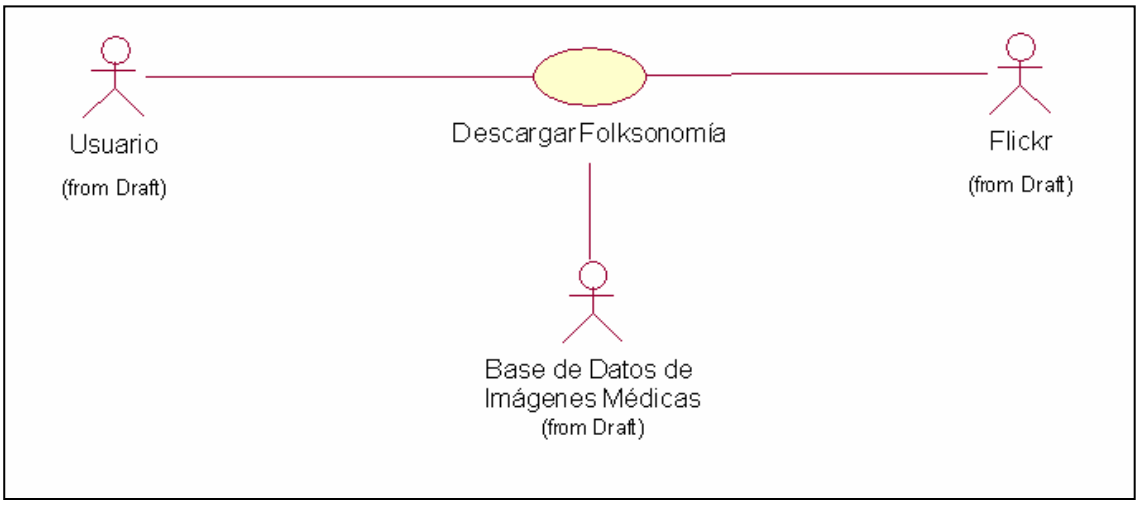


Figura 2.2. Iteración 1 de FlickrBackup. Diagrama de Casos de Uso.

Actores Principales:

El actor principal es el usuario de la aplicación.

Actores Secundarios:

Los actores secundarios son el sistema de *Flickr* y el sistema gestor de bases de datos que contiene la base de datos de imágenes médicas.

Personal Involucrado e Intereses:

Usuario: desea descargar imágenes de *Flickr* y almacenarlas en disco.

Precondiciones:

El usuario debe ejecutar la aplicación y tener una serie de palabras clave a buscar, y la aplicación debe tener conexión con la base de datos.

Poscondiciones:

Se habrán almacenado en disco las imágenes y su información asociada en la base de datos de aquellos recursos de *Fickr* que contengan las etiquetas introducidas como entrada.

Flujo Básico:

1. El usuario inicia la aplicación.
2. Introduce las etiquetas por las que se van a buscar recursos en *Flickr*.
3. Introduce un número máximo de imágenes a descargar, o introduce 0 para no definir un tope.
4. La aplicación busca los identificadores de las imágenes de *Flickr* que cumplen con los requisitos.
5. Se va almacenando cada una de las imágenes y su información asociada.
6. Se cierra la aplicación.

Flujo Alternativo:

No hay flujo alternativo.

2.1.3. Diagramas de Clases.

A continuación se van a mostrar los diagramas de clases de la aplicación. Para mayor claridad, en caso de mostrarse, sólo se han escrito los atributos y métodos más representativos de cada clase.

2.1.3.1. Descargar Folksonomía.

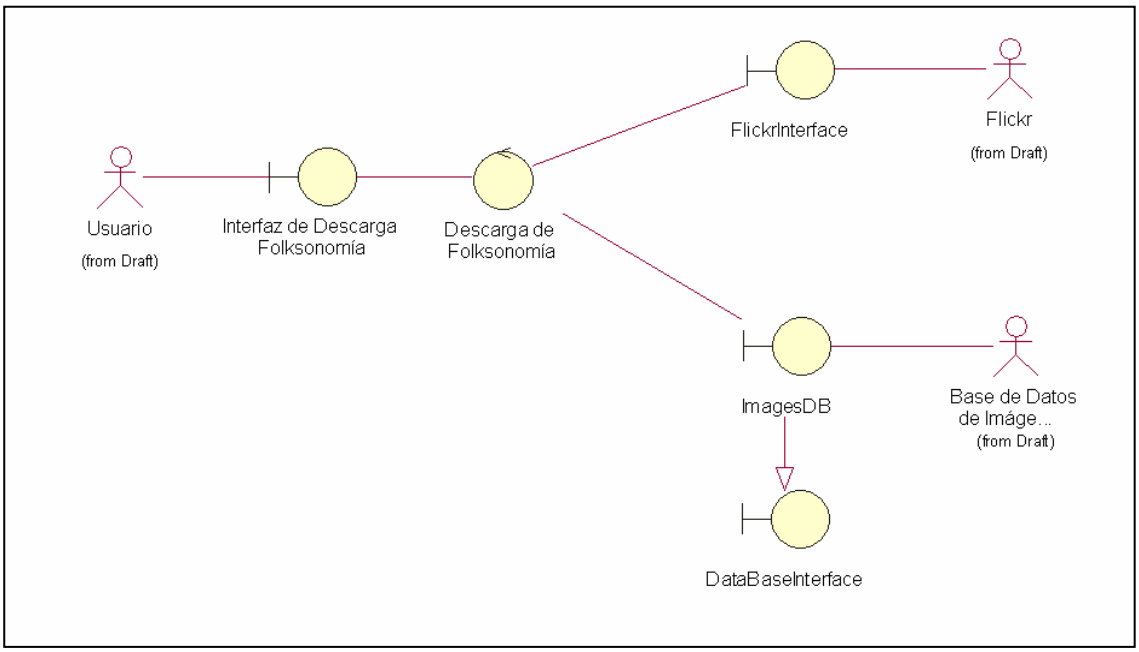


Figura 2.3. Diagrama de clases de FlickrBackup. Descargar Folksonomía.

La figura 2.3 muestra las clases esteriotipadas de la aplicación. Pero debido a que la interfaz de usuario de la misma es muy simple se ha decidido unir ésta con la clase "Descarga de Folksonomía" en una única clase *Main*. A continuación, la figura 2.4, muestra los detalles.

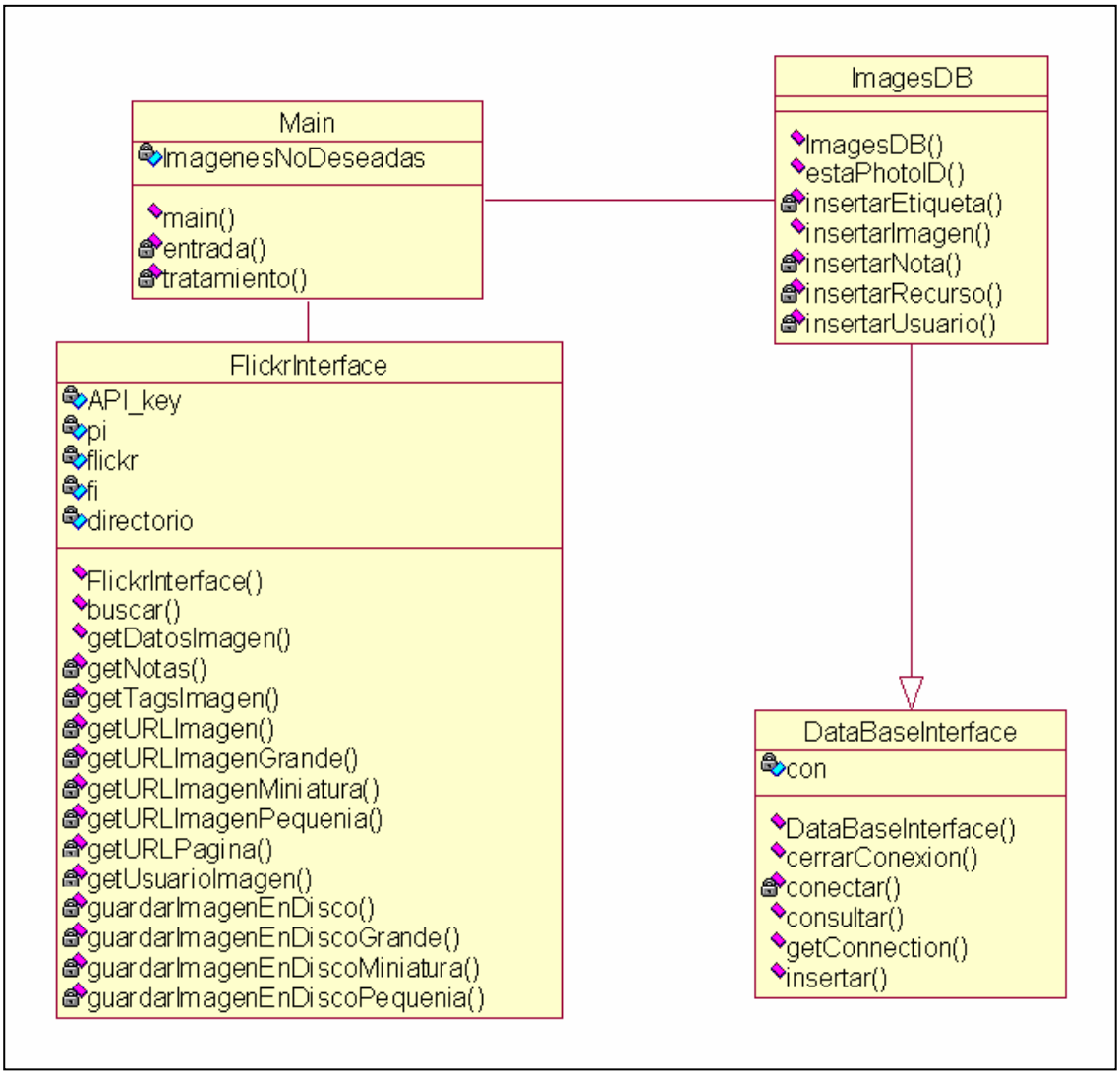


Figura 2.4. Diagrama de Clases de FlickrBackup definitivo.

Como se ha comentado con anterioridad, la clase *Main* se encarga de proporcionar de una interfaz de usuario muy básica, ya que no es necesario nada más, y también se encarga de realizar los cálculos y la computación de la aplicación. Las clases *FlickrInterface* e *ImagesDB* proporcionan métodos para comunicarse con sistemas externos a la aplicación que son necesarios para el funcionamiento de la misma. *FlickrInterface* hace el papel de interfaz con el sistema de *Flickr*. Es esta clase la que hace uso de la biblioteca de clases de java que implementa el *API* de *Flickr*, *flickrj*[1]. *ImagesDB* es la clase intermedia entre la aplicación y el sistema gestor de bases de datos, en este caso *MySQL*, y proporciona métodos concretos para almacenar la información asociada a las imágenes. Además está la superclase *DataBaseInterface* que proporciona unos métodos básicos para comunicarse con el sistema gestor de bases de datos, la cual es heredada por *ImagesDB*.

[1] http://flickrj.sourceforge.net/

2.1.4. Arquitectura del Sistema.

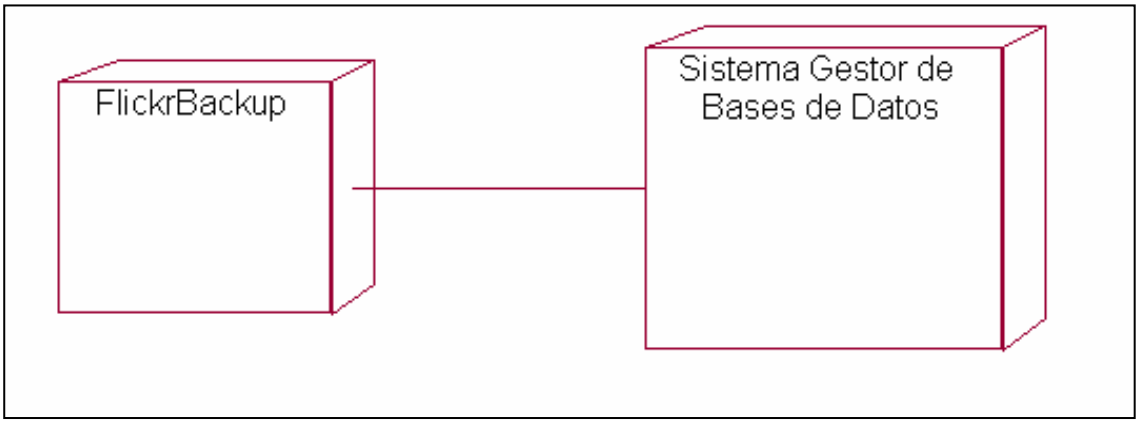


Figura 3.5. Diagrama de Despliegue de FlickrBackup.

La arquitectura de esta aplicación consta de dos nodos. El primer nodo es aquel en que se ejecuta la aplicación en sí. El segundo nodo es que cobija al sistema gestor de bases de datos donde se ha instalado la base de datos de imágenes médicas.

2.1.5. Diagramas de Flujo y de Secuencia.

A continuación se van a mostrar los diagramas de secuencia y de flujo de la aplicación.

2.1.5.1. Diagrama de Flujo.

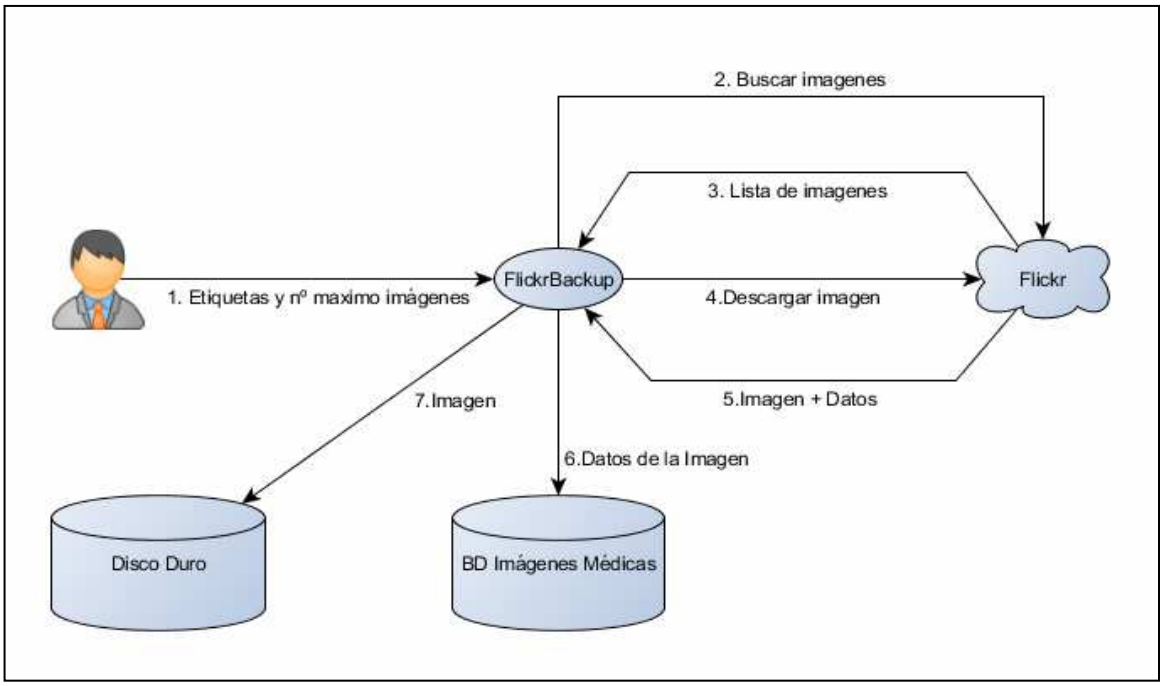


Figura 2.6. Diagrama de Flujo de FlickrBackup.

La figura 2.6 muestra el funcionamiento general de la aplicación *FlickrBackup*. Indica como, primero, el usuario introduce las etiquetas y el número máximo de imágenes a descargar. Seguidamente, la aplicación hace una búsqueda de las imágenes con las etiquetas suministradas por el usuario. Por último, la aplicación procede a descargar las imágenes de la lista, almacenando los datos de la imagen en la base de datos, y el archivo que contiene la imagen en el disco duro.

2.1.5.2. Diagrama de Secuencia. Descargar Folksonomía.

La secuencia de ejecución de la aplicación sería la siguiente. Al iniciarse el programa, éste le pregunta al usuario por las etiquetas por las que buscar imágenes y si desea establecer un número máximo de imágenes a descargar en la ejecución. Después de introducir los parámetros el programa hace una consulta a *Flickr* buscando imágenes que contengan las etiquetas detalladas por el usuario. Entonces *Flickr* devuelve una lista con los identificadores de las imágenes con los requisitos establecidos en la consulta. Por cada uno de estos identificadores, o hasta que se halla llegado al tope establecido, se descargan y almacenan las imágenes, en distintos tamaños, en disco y se obtiene la información asociada de la imagen como puede ser el usuario que la subió, las etiquetas que tiene anotadas y las notas. Después toda esta información es pasada a una instancia de la clase *ImagesDB* para insertarla en la base de datos. Una vez que se han tratado todas las imágenes deseadas la aplicación termina su ejecución.

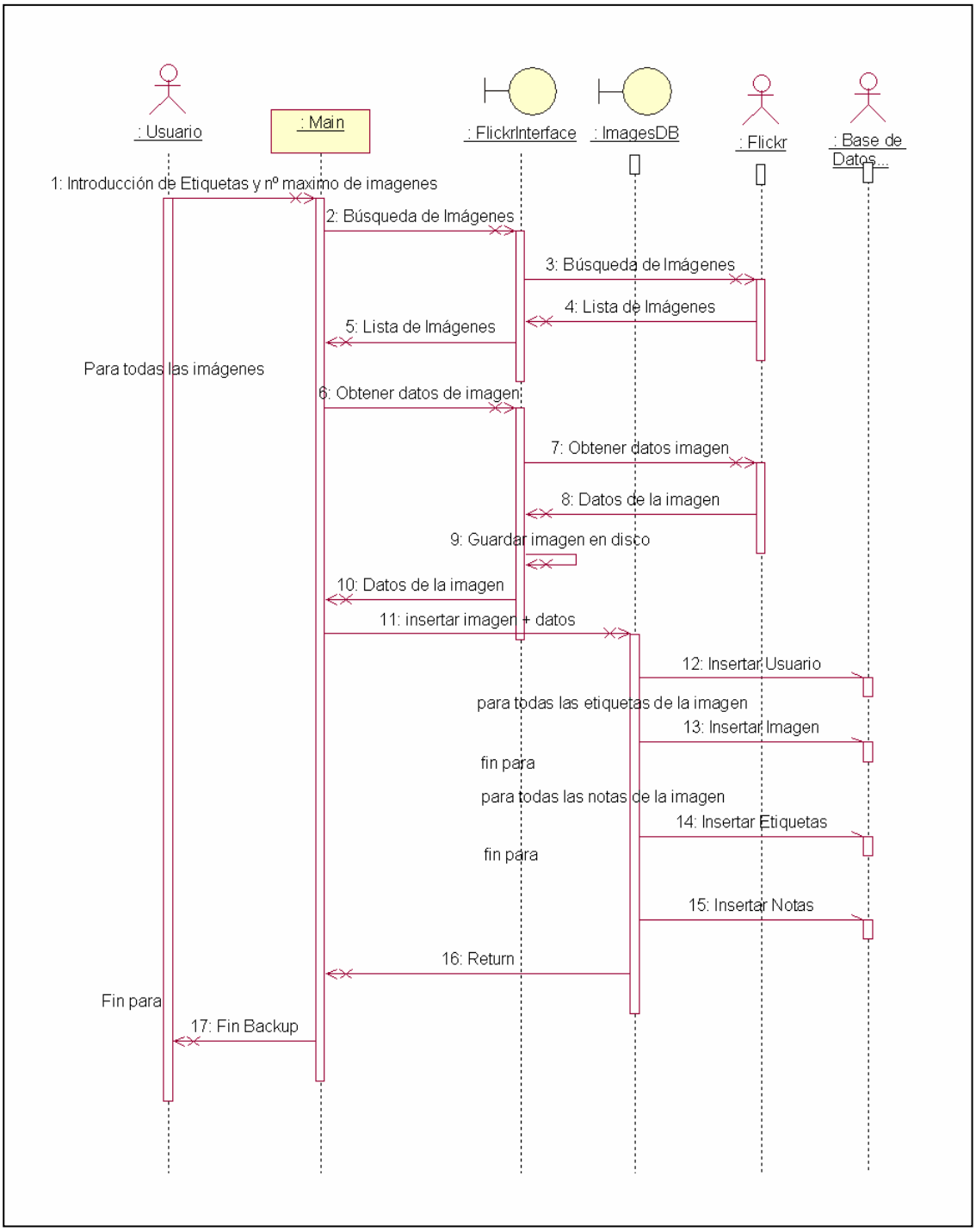


Figura 2.7. Diagrama de Secuencia de FlickrBackup.

2.1.6. Modelo Entidad-Relación y Modelo Relacional.

2.1.6.1. Diagramas Entidad – Relación.

Se ha decidido usar un sistema gestor de bases de datos para almacenar la información asociada a las imágenes descargadas de *Fickr*. Para ello se ha definido una base de datos que recrea una folksonomía.

2.1.6.1.1. Entidad-Relación de la base de datos de imágenes médicas.

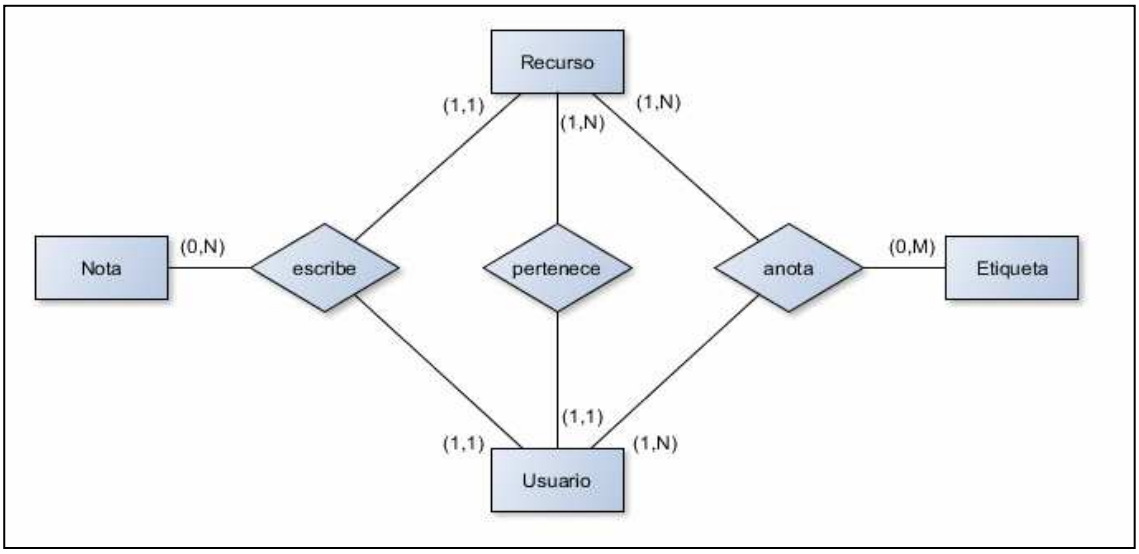


Figura 2.8. Diagrama Entidad-Relación de la base de datos de imágenes médicas.

La base de datos representada por el diagrama entidad-relación de la figura 2.8 pretende contener una folksonomía con recursos (en nuestro caso concreto serían las imágenes médicas), usuarios, etiquetas y notas (las entidades del diagrama). Cada recurso pertenece a uno y sólo un usuario. A su vez, los recursos están anotados por etiquetas que escriben los usuarios. Cada recurso puede estar anotado por múltiples etiquetas y estas, pueden ser escritas por múltiples usuarios también. La entidad "Nota" representa una serie de apuntes que pueden hacer los usuarios de *Flickr* sobre la propia superficie de la imagen. Cada recurso puede tener escritas múltiples notas, pero estas sólo pueden ser escritas por un único usuario y sólo pertenecen a un único recurso. Cada usuario puede escribir múltiples notas.

2.1.6.2. Diseño de la Base de Datos: Modelo Relacional.

Esta sección muestra el diseño de las tablas surgidas a partir del modelo entidad-relación descrito en la sección anterior. La figura 2.9 expone gráficamente el modelo relacional de la base de datos de imágenes médicas.

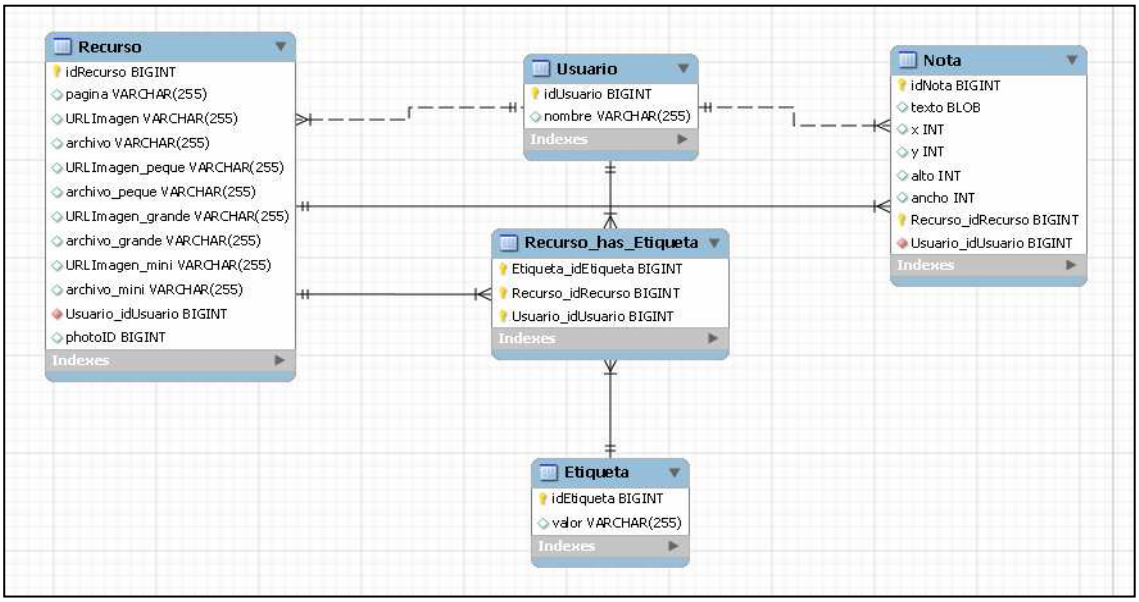


Figura 2.9. Esquema de la base de datos de imágenes médicas.

A continuación se describe cada una de las tablas que contiene la base de datos de imágenes médicas.

Recurso: representa cada una de las imágenes.

- *idRecurso*: es el identificador de la imagen.
- pagina: contiene la *URL* de la página de *Flickr* en la que se aloja la imagen.
- *URLImagen*: contiene la *URL* de la imagen de tamaño mediano.
- *archivo*: almacena la ruta del archivo donde está almacenada la imagen en su versión de tamaño mediano.
- *URLImagen_peque*: contiene la *URL* de la imagen de tamaño pequeño.
- *archivo_peque*: almacena la ruta del archivo que contiene la imagen de tamaño pequeño.
- *URLImagen_grande*: contiene la *URL* de la imagen de tamaño grande.
- *archivo_grande*: almacena la ruta del archivo que contiene la imagen de tamaño grande.
- *URLImagen_mini*: contiene la *URL* de la imagen de tamaño miniatura.
- *archivo_mini*: almacena la ruta del archivo que contiene la imagen de tamaño miniatura.
- *photoID*: es el identificador de la imagen dentro del sistema de *Flickr*.
- *Usuario_idUsuario*: es el identificador del usuario que subió la imagen. Es una referencia externa de la tabla *Usuario*.

Usuario: representa al usuario en el sistema.

- *idUsuario*: es el identificador del usuario.

- *nombre*: es el *nick* o apodo que toma el usuario dentro del sistema.

Nota: contiene las notas que se escriben sobre las imágenes en *Flickr*.
- *idNota*: es el identificador de cada nota.
- *texto*: es el mensaje contenido en la nota.
- *x*: es la coordenada horizontal que define el vértice superior izquierdo del cuadrado que delimita la zona de la imagen que señala la nota.
- *y*: es la coordenada vertical que define el vértice superior izquierdo del *cuadrado* que delimita la zona de la imagen que señala la nota.
- *alto*: almacena la altura del cuadrado.
- *ancho*: almacena la achura del cuadrado.
- *Recurso_idRecurso*: almacena el identificador del recurso donde está escrita dicha nota.
- *Usuario_idUsuario*: almacena el identificador del autor de dicha nota.

Etiqueta: contiene las etiquetas usadas para hacer anotaciones en las imágenes.
- *idEtiqueta*: es el identificador de cada etiqueta.
- *valor*: es la cadena de texto que da valor semántico a la etiqueta.

Recurso_has_Etiqueta: es la tabla que relaciona los recursos con las etiquetas.
- *Etiqueta_idEtiqueta*: es el identificador de la etiqueta.
- *Recurso_idRecurso*: es el identificador del recurso.
- *Usuario_idUsuario*: es el identificador del usuario que anotó dicha etiqueta en el recurso.

2.2. Análisis y Diseño de *ACoAR*.

Una vez obtenida la folksonomía de imágenes médicas podemos pasar al desarrollo del método que haga la clasificación automática de la misma. Como ya se ha comentado con anterioridad, para ello se va a hacer uso de parte del trabajo hecho por Francisco Echarte en su tesis doctoral [4]. Él propone un método llamado *ACoAR* (*Automatic Classification of Annotated Resources*). Este método es capaz de crear y actualizar una estructura de datos que contiene toda la información de los recursos (R), los conceptos (C) y la clasificación de los recursos en dichos conceptos (Z), entre otras cosas. Esta creación y actualización la hace en base a una folksonomía y los cambios que, con el tiempo, se producen en ella. De forma de acotar la extensión y duración de este proyecto, de todo el método propuesto por Francisco Echarte sólo se va a hacer uso de la parte que se encarga de la creación de la estructura de datos.

Debido a nos basamos en el trabajo ya desarrollado en la tesis no se va a exponer en detalle el análisis y diseño, a excepción de aquellas partes que hallan podido variar en la implementación del método, porque esto ya fue realizado por Francisco Echarte.

En esta sección se va a tratar el análisis y diseño de la aplicación *ACoAR*, centrándose más en los aspectos que difieren del propio diseño que Francisco Echarte presenta en su tesis [4] y siempre ajustándonos a la parte de creación del modelo.

2.2.1. Análisis de Requisitos

A continuación se van a definir los requisitos funcionales y no funcionales de la parte de *ACoAR* que se va a tratar en este proyecto.

2.2.1.1. Requisitos Funcionales.

- Debe, a partir de una folksonomía, construir una serie de conceptos y usarlos para clasificar los recursos de la misma.
- Debe poder construir tablas de similaridades entre los diferentes elementos del modelo.

2.2.1.2. Requisitos No Funcionales.

- La aplicación resultante debe estar construida de tal forma que facilite lo más posible su portabilidad de una máquina a otra.
- Los resultados de ejecución de la aplicación deben persistir en el tiempo. Deben ser almacenados en algún lugar, como en una base de datos.
- La aplicación debe tener un control lo más eficiente posible de la memoria, debido a que va a tener que tratar con gran cantidad de información.
- Debe ser construida de una forma modular, de forma que sea lo más fácil posible el intercambiar los algoritmos, métodos y fórmulas que definen cada una de las partes del algoritmo de creación del modelo.

2.2.2. Casos de Uso.

Esta parte del ciclo de vida no se a estudiar debido a que se considera que ya fue tratada por Francisco Echarte.

2.2.3. Diagramas de Clases.

Este apartado va a mostrar ciertos diagramas de clases sobre el método de creación del modelo de datos de *ACoAR*. Sólo se van a mostrar aquellos diagramas de clases que muestren variaciones con respecto del diseño original de Francisco Echarte o que correspondan a partes de la aplicación propias de la implementación hecha en este proyecto.

2.2.3.1. Estructura principal *ACoAR*. Creación de *ModelACoAR*.

La figura 2.10 presenta el diagrama de clases de la estructura principal de *ACoAR*. Prácticamente el diseño de estas clases es casi idéntico al presentado en [4], sólo que en este caso se han omitido ciertos elementos que correspondían a otros métodos diferentes a la creación del modelo, y se han podido o cambiar otros electos por diversas circunstancias. Puede verse que hay una clase principal, *ACoAR*, que hace uso de todas las demás. El resto de clases, o son interfaces o clases abstractas, o son clases que heredan o implementan a las anteriores. Esta estructura fue diseñada de forma que *ACoAR* utilizase cada módulo del algoritmo por medio de las interfaces y las clases abstractas, haciendo de esta forma que para cambiar el algoritmo, método u fórmula usado por uno de los componentes tan sólo habría que crear otra clase que heredase la correspondiente clase abstracta o implementase la interfaz, consiguiendo que este cambio no afectase al resto de la implementación.

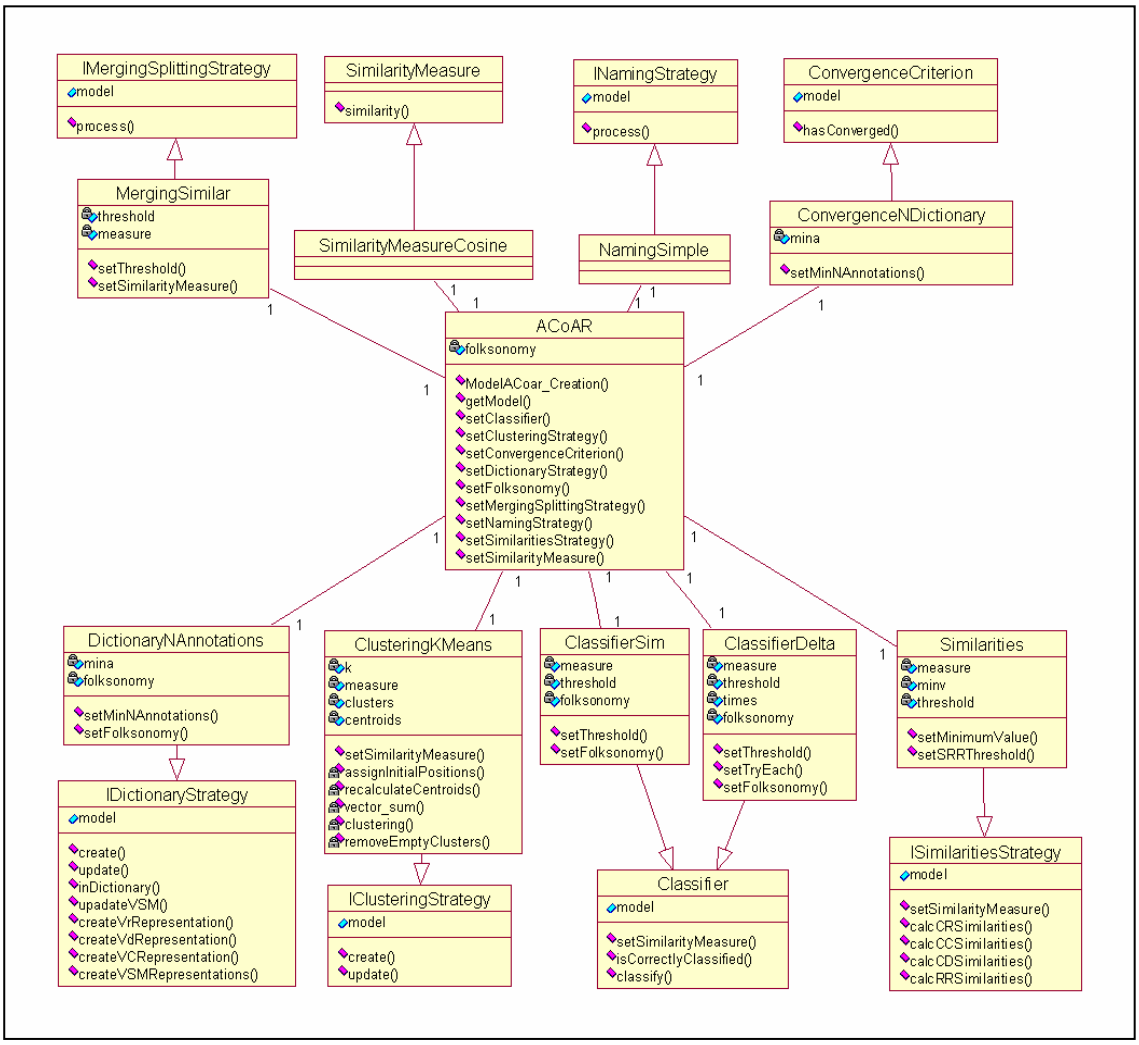


Figura 2.10. Diagrama de Clases de la estructura principal de ACoAR.

A continuación vamos a explicar la funcionalidad de cada una de las clases[1]. Primero vamos a definir las interfaces y clases abstractas, recordemos que estas sólo van a definir los métodos, y ciertos atributos, que luego tendrán que implementarse en las clases que las

[1] Las descripciones están apoyadas por las definiciones dadas por Francisco Echarte en su tesis doctoral y/o diversas publicaciones.

hereden. En la implementación hecha en este proyecto todos estos componentes han sido definidos como clases abstractas:

- *IDictionaryStrategy:* proporciona métodos para crear y actualizar el diccionario, los vectores representativos de los recursos, etiquetas y conceptos.
- *IClusteringStrategy:* se encarga de agrupar los recursos creando o actualizando los conceptos y la clasificación de los recursos en estos (Z).
- *Classifier:* se utiliza para clasificar los recursos bajo los conceptos.
- *ISimilaritiesStrategy:* se encarga de calcular las medidas de similaridad entre diferentes elementos del modelo (concepto-recurso, concepto-concepto, concepto-diccionario, recurso-recurso).
- *IMergingSplittingStrategy:* se encarga de unir o dividir los conceptos cuando se cumpla una determinada condición, con objeto de afinar el resultado obtenido por *IClusteringStrategy*.
- *SimilarityMeasure:* define la medida de similaridad utilizada.
- *INamingStrategy:* se encarga de asignar un nombre a cada concepto.
- *ConvergenceCriterion:* proporciona métodos para indicar cuando han convergido las anotaciones de un recurso.

Pero todas estas clases no sirven para nada si no hay otras que las implementen. Seguidamente se van a explicar las clases que heredan de la anteriormente expuestas nombrando el método que utilizan para definir la funcionalidad.

- *DictionaryNAnnotations:* Hereda de *IDictionaryStrategy*. Construye el diccionario escogiendo aquellas etiquetas que su número de anotaciones supere un mínimo. A partir de este diccionario se encarga de construir los vectores de representación de los recursos, etiquetas y conceptos.
- *ClusteringKMeans:* Hereda de *IClusteringStrategy*. Define los conceptos y la clasificación de los recursos en dichos conceptos mediante el algoritmo *KMeans*.
- *ClassifierSim:* Hereda de *Classifier*. Define el clasificador de tipo *sim*.
- *ClassifierDelta:* Hereda de *Classifier*. Define el clasificador de tipo delta.
- *Similarities:* Hereda de *ISimilaritiesStrategy*. Calcula la similaridad de todos los elementos entre sí para las comparaciones concepto-concepto, concepto-recurso, concepto-diccionario. Para las comparaciones recurso-recurso sólo calcula la similaridad entre recursos de un mismo concepto y entre recursos de conceptos cuya similaridad supere un cierto umbral.
- *MergingSimilar:* Hereda de *IMergingSplittingStrategy*. Une los conceptos cuya similaridad supere un cierto umbral.
- *SimilarityMeasureCosine:* Hereda de *SimilarityMeasure*. Implementa la medida de similaridad del coseno.
- *NamingSimple:* Hereda de *INamingStrategy*. Crea el nombre del concepto concatenado el texto de las etiquetas más representativas del vector que lo define.
- *ConvergenceNDictionary:* Hereda de *ConvergenceCriterion*. Da como convergido a un recurso cuando la suma de las componentes de su vector representativo es mayor o igual a cierto umbral.

Por último nos queda explicar la clase principal *ACoAR*. Esta clase proporciona una serie de métodos para introducir las instancias de cada uno de los módulos que componen la aplicación. Además tiene el método *ModelACoAR_Creation()* que es el que se encarga de la creación del modelo haciendo uso de todas las clases mencionadas anteriormente. Para

ver de una forma más detallada el funcionamiento de este método se expone en la figura 3.14 su diagrama de secuencia.

2.2.3.2. Modelo de datos de *ACoAR*. *ModelACoAR*.

Esta sección va a explicar las clases que se ocupan de almacenar los datos generados en la ejecución de *ACoAR*. La figura 2.11 muestra el diagrama de clases del modelo de datos de *ACoAR*.

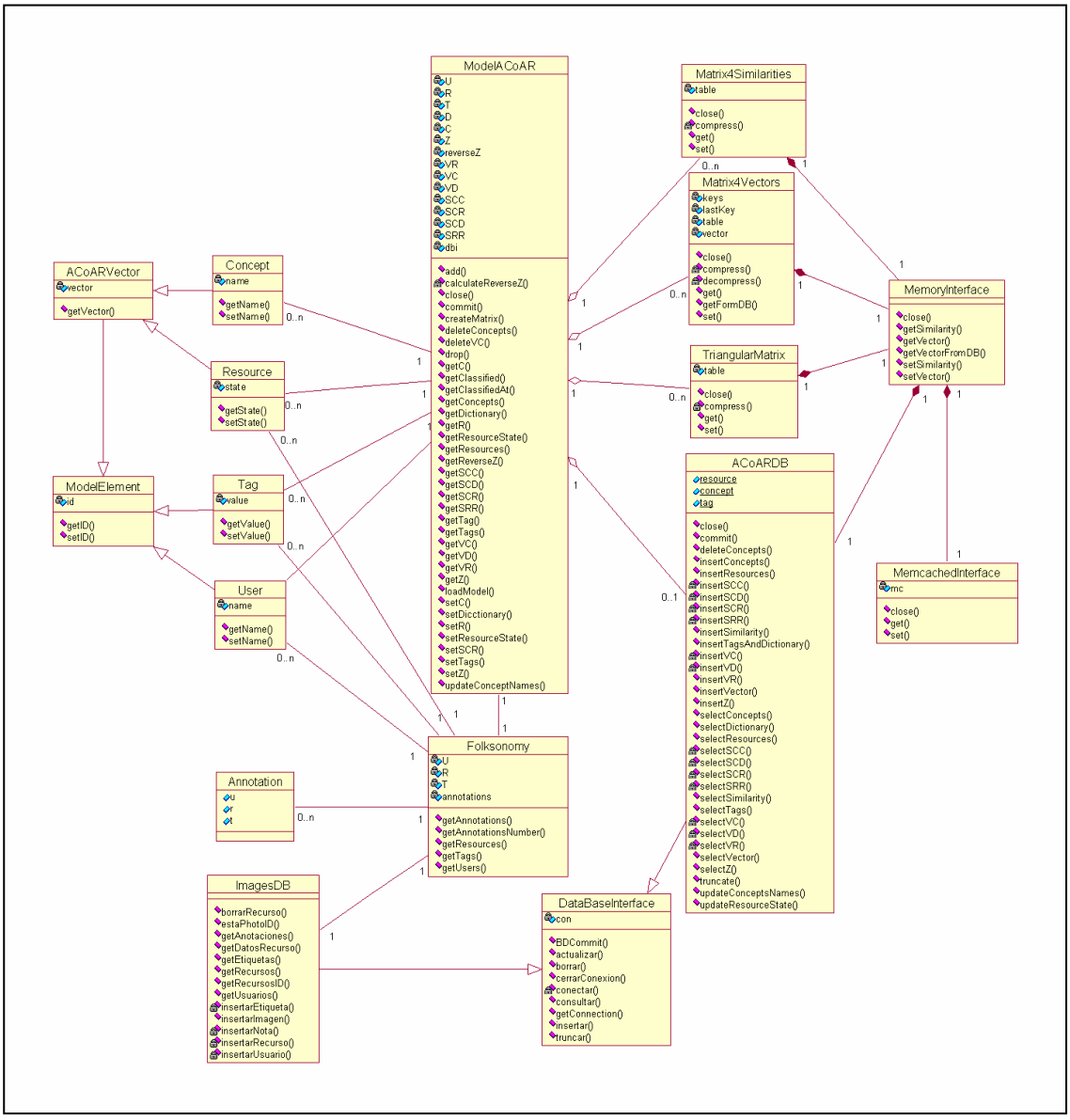


Figura 2.11. Diagrama de clases del modelo de datos de ACoAR.

Como puede observarse este modelo tiene una clase principal, *ModelACoAR* que es la que hace de interfaz del modelo, además de almacenar ciertos datos. Los atributos de la clase se encargan de guardar los datos del modelo, y los métodos constituyen una interfaz para

acceder a ellos. A continuación vamos a explicar qué es lo que almacena cada atributo, también decir que algunos de ellos están definidos con clases que se explicarán a posteriori:

- *U*: Almacena el conjunto de usuarios. Es una tabla *hash* de objetos de la clase *User* que toma el identificador de usuario como clave.
- *R*: Almacena el conjunto de recursos. Es una tabla *hash* de objetos de la clase *Resource* que toma el identificador de recurso como clave.
- *T*: Almacena el conjunto de etiquetas. Es una tabla *hash* de objetos de la clase *Tag* que toma el identificador de etiqueta como clave.
- *D*: Almacena el diccionario. Es una lista con los identificadores de las etiquetas que pertenecen al diccionario.
- *C*: Almacena el conjunto de conceptos. Es una tabla *hash* de objetos de la clase *Concept* que toma el identificador de concepto como clave.
- *Z*: Almacena la clasificación de los recursos. Es una tabla *hash* que toma como clave el identificador del recurso y como valor el identificador del concepto. Nos indica a que concepto pertenece cada recurso.
- *reverseZ*: También almacena la clasificación de los recursos pero en sentido inverso. Nos dice que recursos pertenecen a cada concepto. Está precalculado para ahorrar tiempo de ejecución. Es una tabla *hash* con el identificador de conceptos como clave y una lista de identificadores de recursos como valor.
- *VR*: Almacena los vectores representativos de los recursos. Es un objeto de la clase *Matrix4Vectors*.
- *VC*: Almacena los vectores representativos de los conceptos. Es un objeto de la clase *Matrix4Vectors*.
- *VD*: Almacena los vectores representativos de las etiquetas del diccionario. Es un objeto de la clase *Matrix4Vectors*.
- *SCC*: Almacena las similaridades concepto-concepto. Es un objeto de la clase *Matrix4Similarities*.
- *SCR*: Almacena las similaridades concepto-recurso. Es un objeto de la clase *Matrix4Similarities*.
- *SCD*: Almacena las similaridades concepto-diccionario. Es un objeto de la clase *Matrix4Similarities*.
- *SRR*: Almacena las similaridades recurso-recurso. Es un objeto de la clase *TriangularMatrix*.
- *dbi*: guarda una instancia de *ACoARDB*.

Otras clases importantes son las que definen a los recursos, usuarios, etiquetas y conceptos. Estas son *Resource*, *User*, *Tag* y *Concept* respectivamente. Estas a su vez heredan de *ACoARVector* o de *ModelElement*, que proporcionan métodos básicos a cada una de ellas. A continuación se describe cada una de estas clases:

- *ModelElement*: Es la clase más básica. Contiene un atributo para almacenar el identificador y proporciona métodos para su lectura y escritura.
- *ACoARVector*: Hereda de *ModelElement*. Tiene un atributo para almacenar el vector representativo del elemento, y proporciona métodos para leerlo y escribirlo. Al final no se ha dado uso a estos elementos ya que se ha decidido almacenar los vectores de otra forma.
- *Resource*: Hereda de *ACoARVector*. Tiene un atributo para almacenar su estado. Además proporciona métodos para la lectura y escritura de dicho atributo.

- *User*: Hereda de *ModelElement*. Define un atributo para almacenar el nombre del usuario. Proporciona métodos de lectura y escritura del mismo.
- *Tag*: Hereda de *ModelElement*. Contiene un atributo para almacenar el valor textual de la etiqueta y proporciona métodos para su lectura y escritura.
- *Concept*: Hereda de *ACoARVector*. Tiene un atributo para guardar el nombre del concepto y proporciona métodos para su lectura y escritura.

En el diagrama de clases también se muestran una serie de clases que permiten la conectividad con un sistema gestor de bases de datos, concretamente a *MySQL*. Hay tres definidas a tal fin, *DataBaseInterface*, *ImagesDB* y *ACoARDB*. La primera es un clase que proporciona los métodos básicos de conexión, cierre, consulta, inserción y borrado. Las otras dos clases heredan de la primera y definen métodos concretos para conectarse con las dos bases de datos que forman parte del sistema, la base de datos de imágenes médicas y la base de datos de *ACoAR*.

A parte de la conectividad con el sistema gestor de bases de datos también es necesario poder establecer conexión con otro sistema externo, *Memcached*[1]. *Memcached* es un sistema distribuido de memoria caché. Se ha usado en este proyecto como capa intermedia entre la base de datos y la aplicación. Para acceder a este sistema se ha diseñado la clase *MemcachedInterface* que hace uso del API *spymemcached*[2].

La clase MemoryInterface tiene como objetivo el coordinar las interacciones con la base de datos y *memcached*. Solamente va a tratar con vectores y con similaridades. Todas las inserciones que le lleguen van a ir directamente a la base de datos. En cambio, cuando se consulte un vector o una similaridad, primero se va a mirar si está indexado en *memchaded*. Si lo está se devuelve el valor, sino, se busca en la base de datos. Si lo encuentra, se indexa el valor en *memcached*, para futuras búsquedas, y se devuelve el valor.

De esta última clase, van a hacer uso otras tres. Estas son *Matrix4Vectors*, *Matrix4Similarities* y *TriangularMatrix*. Dichas clases van a ser las encargadas de gestionar los vectores representativos y las similaridades. *Matrix4Vectors* se va a encargar de los vectores y *Matrix4Similarities* de las similaridades. Pero *TriangularMatrix* se va a encargar concretamente de las similaridades recurso-recurso. Esta clase representa una matriz triangular, es decir, que solamente se almacenan los datos de la diagonal de la matriz y los que quedan por encima, o los datos de la diagonal de la matriz y los que quedan por debajo. Guardamos solamente la mitad de los datos porque al calcular las similaridades recurso-recurso nos va a quedar una matriz cuadrada cuya parte triangular superior va a ser igual que la inferior. Así ahorramos espacio ya que normalmente esta es una matriz con gran cantidad de datos. Además la clase *TriangularMatrix* está construida de tal forma que te simula las dos matrices triangulares, tanto la superior como la inferior. Por tanto no importa el orden de los índices a la hora de hacer consultas ni inserciones.

Otra importante es la de *Folksonomy*, que representa la folksonomía que se toma como datos de entrada del algoritmo. Almacena los usuarios, recursos y etiquetas como *ModelACoAR*, pero además tiene una lista de objetos *Annotation*. Esta clase no es más que una tupla usuario-recurso-etiqueta que viene a representar las anotaciones de los recursos.

[1] http://memcached.org/
[2] http://code.google.com/p/spymemcached/

2.2.4. Arquitectura del Sistema.

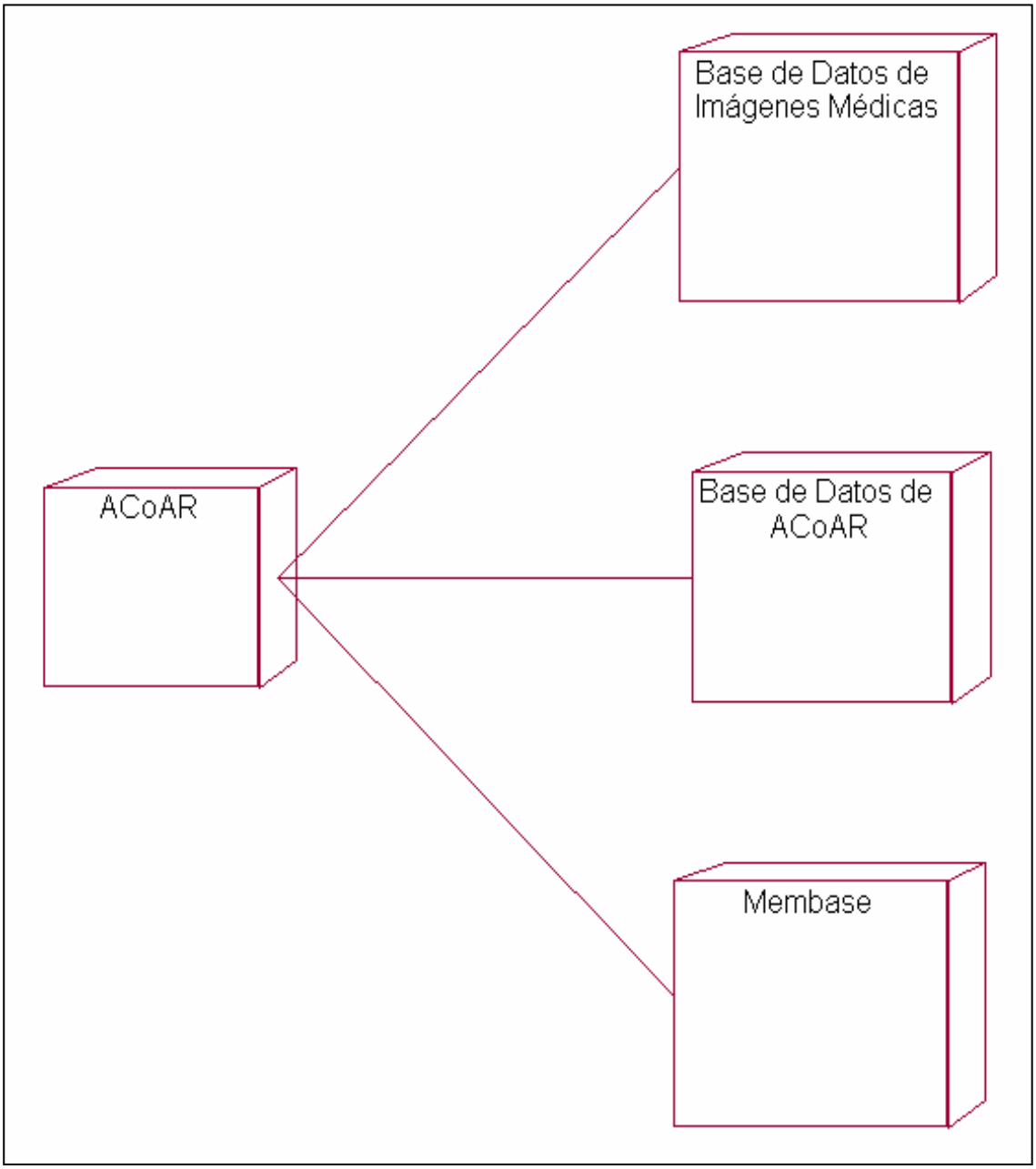


Figura 2.12. Diagrama de Despliegue de ACoAR.

Como muestra la figura 2.12, la arquitectura de esta implementación de *ACoAR* consta de cuatro nodos. El primer nodo es aquel en que se ejecuta la aplicación *ACoAR* en sí. A su vez, éste hace uso de los otros tres nodos. Hay dos que son los que contienen las bases de datos. Por un lado está, la base de datos de imágenes médicas, y por el otro, la base de datos de *ACoAR*. El cuarto y último nodo es el que tiene el sistema de memoria caché *Membase*.

2.2.5. Diagramas de Flujo y de Secuencia.

Este apartado va a mostrar ciertos diagramas de flujo y de secuencia sobre el método de creación del modelo de datos de *ACoAR*. Sólo se van a mostrar aquellos diagramas que muestren variaciones con respecto del diseño original de Francisco Echarte o que correspondan a partes de la aplicación propias de la implementación hecha en este proyecto.

2.2.5.1. Diagrama de Flujo.

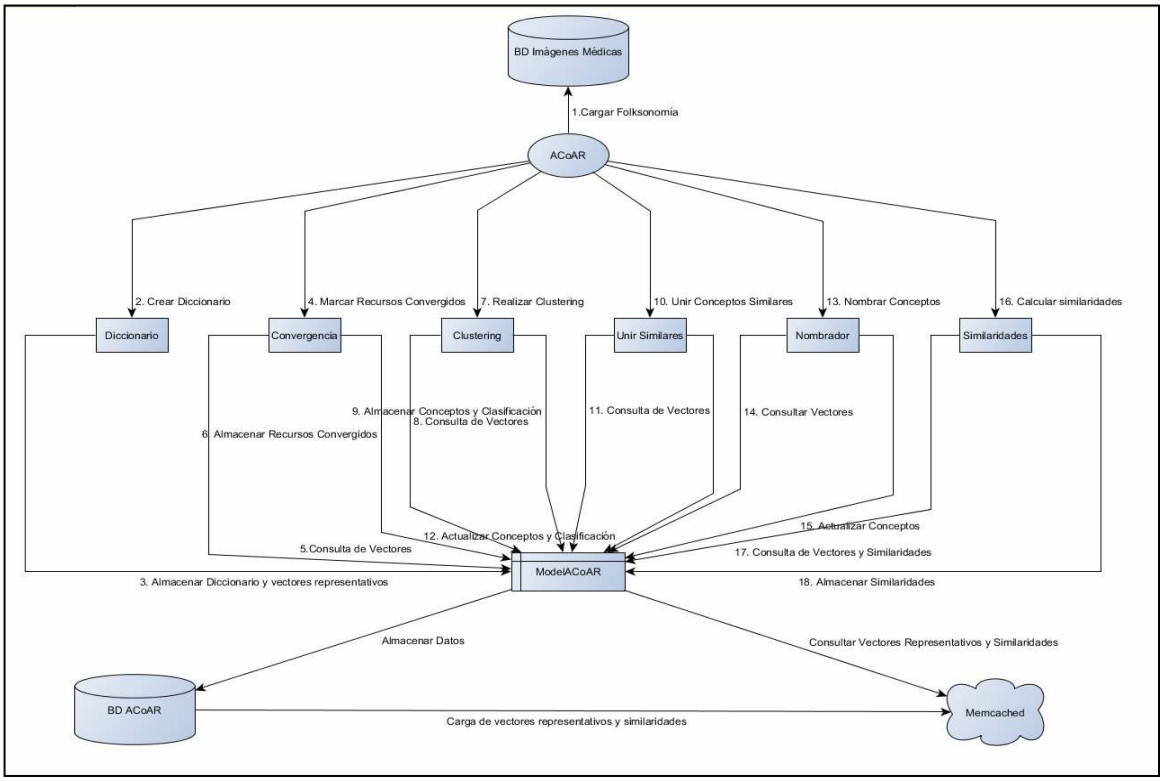


Figura 2.13. Diagrama de Flujo de ACoAR.

La figura 2.13 muestra el comportamiento general de la aplicación. Puede verse como se ejecutan de forma secuencial los módulos del diccionario, de convergencia, de *clustering*, de unión de conceptos, de nombrado de conceptos y de cálculo de similaridades. También aparece como cada uno de estos componentes consulta a *ModelACoAR* sobre los vectores representativos y similaridades cuando les es necesario, y piden que almacene los datos que generan. A su vez, sale reflejado en la figura como se comunica *ModelACoAR* con *Memcached* para consultar los vectores y las similaridades, y como lo hace con la base de datos para almacenar la información generada por el sistema.

2.2.5.2. Diagrama de Secuencia. Estructura principal *ACoAR*. Creación de *ModelACoAR*.

En el diagrama de secuencia, figura 2.14, puede verse como la clase *ACoAR* interacciona con las demás para crear el modelo. La primera acción que realiza es crear el diccionario haciendo una llamada a la clase *DictionaryNAnnotations*. En esa misma llamada, se crean los vectores representativos de los recursos y de las etiquetas del diccionario. Después, usando la clase *ConvergenceNDictionary* decide cuales de los recursos convergen y cuales no, clasificándolos con las etiquetas *converged* y *pending* respectivamente. Una vez hecho esto, pasa los recursos considerados como convergidos a la clase *ClusteringKMeans* para llevar a cabo la creación de los conceptos, definidos por sus vectores de representación, y la clasificación de los recursos en base a estos. Esta clase se apoya a su vez en la clase *SimilarityMeasureCosine* para realizar cálculos de similaridad entre los vectores representativos de los recursos y los conceptos. Tras realizar el *clustering*, el algoritmo procede a realizar la unión de conceptos similares llamando a la clase *MergingSimilar*. A consecuencia de este paso se han producido cambios en la clasificación de los recursos, por tanto, los vectores de los conceptos ya no son representativos de los recursos que contienen. Debido a esto se vuelve a llamar a la clase *DictionaryNAnnotations* para que recalcule los vectores de los conceptos. Una vez hecho esto, el siguiente paso es comprobar si los recursos están correctamente clasificados en los conceptos que les corresponde. Para ello se hace uso de la clase *ClassifierDelta* o *ClassiferSim*, dependiendo del número de conceptos en el sistema (*ClassifierDelta* necesita que por lo menos haya dos, en cambio *ClassifierSim* sólo necesita que haya uno) o de si se ha elegido *ClassifierSim* como clasificador por defecto. A su vez, estas dos clases hacen uso de *SimilarityMeasureCosine* para medir la similaridad entre los recursos y los conceptos. Este proceso de comprobar la correcta clasificación de los recursos conlleva en hecho de eliminar aquellos mal clasificados del domino del concepto, por tanto puede darse el caso de que nos encontremos con conceptos vacíos. A consecuencia de esto, seguidamente, se procede a borrar aquellos conceptos que hubiesen quedado vacíos y a recalcular los vectores de los conceptos. Una vez que se tienen los conceptos y la clasificación bien definida, se llama a la clase *NamingSimple* para dar nombre a los conceptos. Por último sólo queda hacer uso de la clase Similarities para realizar los cálculos de similaridad entre los diferentes elementos del modelo. Esta clase también hace uso de *SimilarityMeasureCosine* para el cálculo de las diversas similaridades.

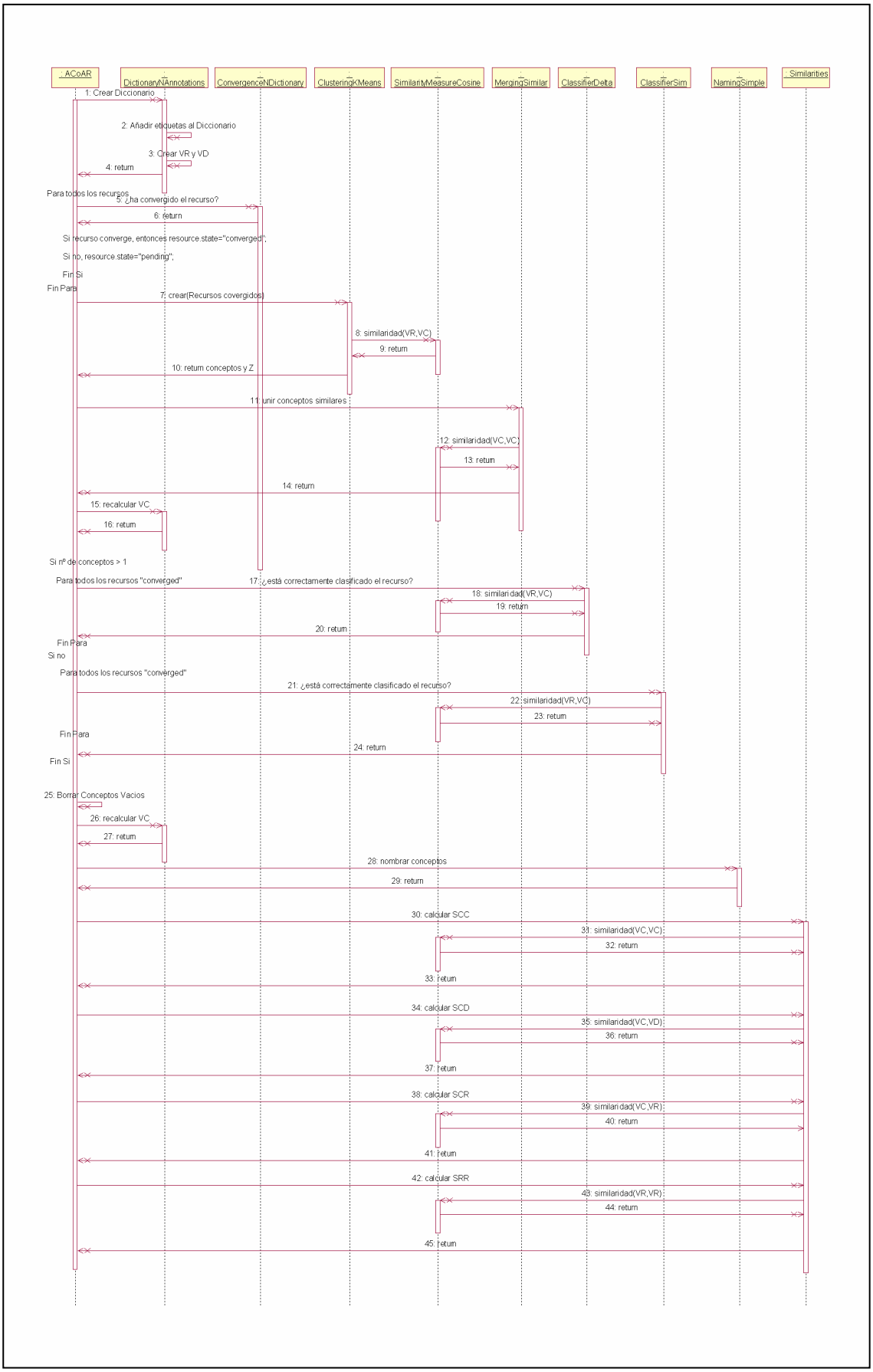


Figura 2.14. Diagrama de Secuencia de ModelACoAR_Creation().

2.2.5.3. Modelo Entidad-Relación y Modelo Relacional.

2.2.6. Modelo Entidad-Relación y Modelo Relacional.

La aplicación a desarrollar necesita que los resultados de su ejecución persistan en el tiempo. Por ello se ha decidido que la mejor forma de que lo hagan es almacenándolos en una base de datos. A continuación se va a exponer el diagrama entidad-relación de la base de datos diseñada para almacenar los datos del modelo generado por *ACoAR*.

2.2.6.1. Diagrama Entidad-Relación de *ACoARDB*.

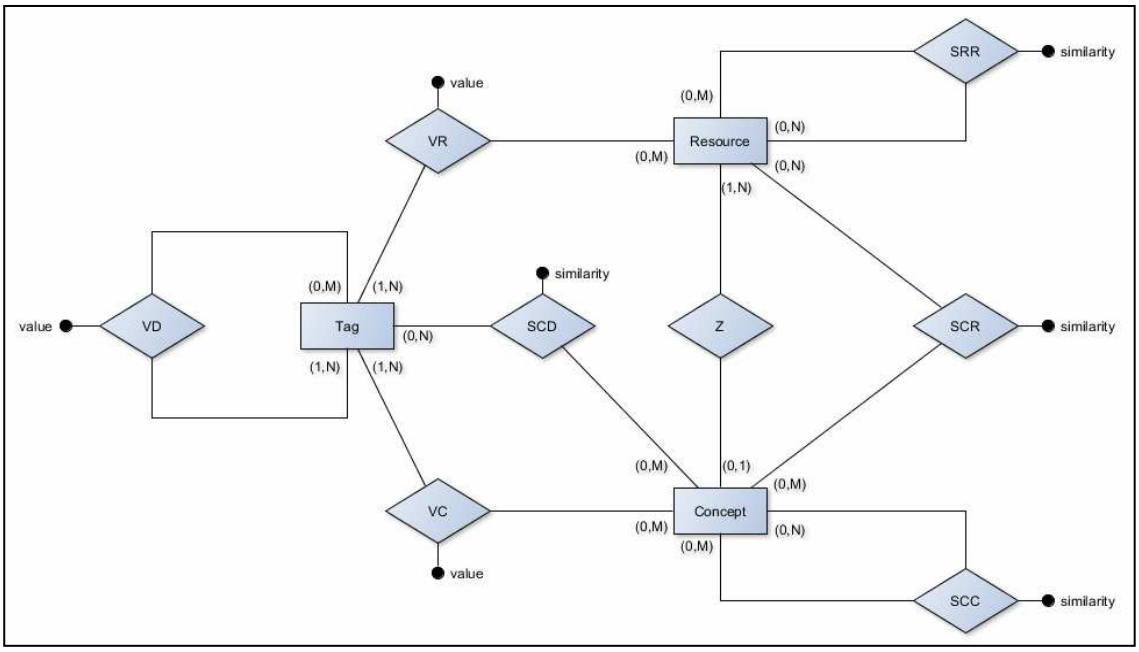


Figura 2.15. Diagrama Entidad-Relación de ACoARDB.

El diagrama entidad-relación de la figura 2.15 representa a la base de datos que va a almacenar el modelo generado por *ACoAR* como datos de salida. Vemos que consta de tres entidades, *Resource* que representa a los recursos de la folksonomía tomada como datos de entrada, *Tag* que almacena las etiquetas anotadas en las imágenes, y *Concept* que va a guardar los conceptos construidos durante la ejecución de la aplicación. Las relaciones que se producen entre las entidades sirven para representar y almacenar los vectores de representación de recursos (*VR*), del diccionario (*VD*) y de conceptos (*VC*), y las similaridades entre las entidades, recurso-recurso (*SRR*), concepto-recurso (*SCR*), concepto-diccionario (*SCD*) y concepto-concepto (*SRR*). Además, de esas relaciones van a salir una serie de atributos que van a guardar valores fruto de dichas relaciones. Las relaciones de los vectores generan un atributo llamado *value* que se va a encarga de almacenar el número de anotaciones de la etiqueta participante en la relación, y las relaciones de las similaridades generan el atributo *similarity* en el cual se va a almacenar el valor de la similaridad de dicha relación.

2.2.6.2. Diseño de la Base de Datos de *ACoAR*. Modelo Relacional.

En esta sección se van a explicar las tablas surgidas del modelo entidad-relación propuesto en la sección anterior. La figura 2.16 muestra gráficamente el modelo relacional de la base de datos de *ACoAR*.

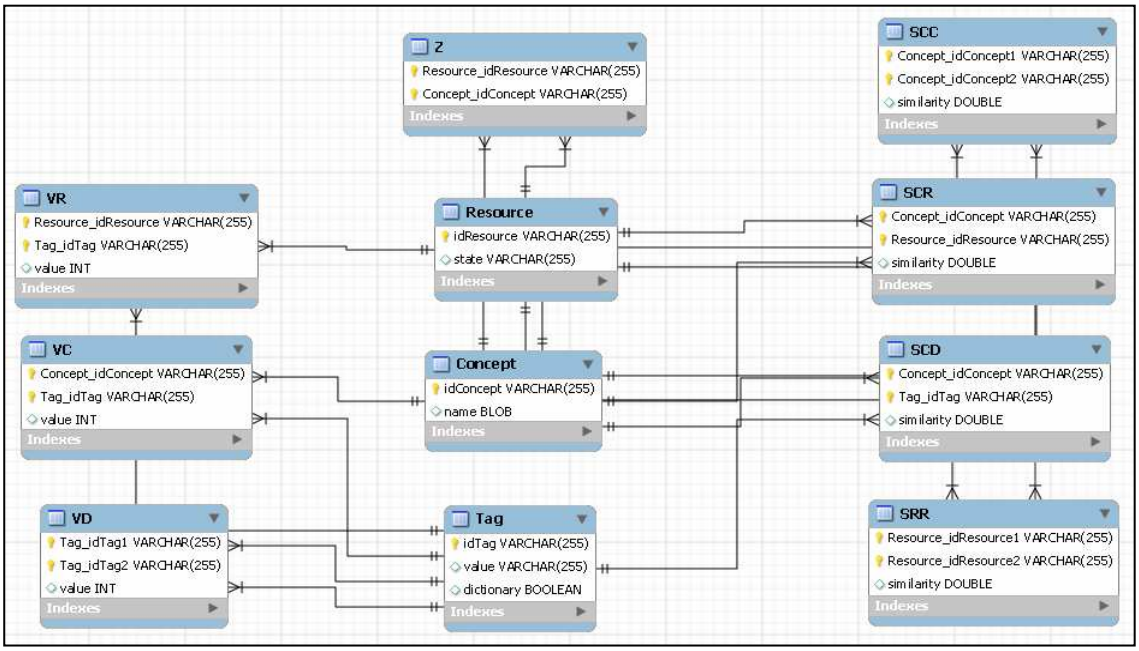


Figura 2.16. Esquema de la base de datos de ACoAR.

Seguidamente se detalla cada una de las tablas de la base de datos de *ACoAR*.

Resource: almacena los recursos del sistema.

- *idResource*: es el identificador del recurso.
- *state*: es el estado del recurso.

Tag: guarda las etiquetas.

- *idTag*: es el identificador de la etiqueta.
- *value*: es el valor textual de la etiqueta.
- *dictionary*: indica si la etiqueta pertenece o no al diccionario.

Concept: guarda los conceptos generados en la aplicación.

- *idConcept*: es el identificador del concepto.
- *name*: es el nombre que toma el concepto.

Z: almacena la clasificación de los recursos.

- *Resource_idResource*: es el identificador del recurso.
- *Concept_idConcept*: es el identificador del concepto en el que está clasificado el recurso.

VR: guarda los vectores representativos de los recursos.

- *Resource_idResource*: es el identificador del recurso.

- *Tag_idTag*: es el identificador de la etiqueta, en este caso perteneciente al diccionario.
- *value*: es el número de anotaciones de la etiqueta en el recurso.

VD: guarda los vectores representativos de las etiquetas del diccionario.
- *Tag_idTag1*: es un identificador de etiqueta que pertenece al diccionario.
- *Tag_idTag2*: es otro identificador de etiqueta perteneciente al diccionario, pudiendo ser el mismo que el de *Tag_idTag1*.
- *value*: es el valor que toma el componente del vector.

VC: guarda los vectores representativos de los conceptos.
- *Concept_idConcept*: es el identificador del concepto.
- *Tag_idTag*: es el identificador de la etiqueta. Pertenece al diccionario.
- *value*: valor que toma el componente del vector. Es la suma de todas las anotaciones de la etiqueta en todos los recursos pertenecientes al concepto.

SCC: guarda las medidas de similaridad de tipo concepto-concepto.
- *Concept_idConcept1*: es un identificador de concepto.
- *Concept_idConcept2*: es otro identificador de concepto. Puede ser el mismo que *Concept_idConcept1*.
- *similarity*: es el valor de la similaridad entre los dos conceptos.

SCR: guarda las medidas de similaridad de tipo concepto-recurso.
- *Concept_idConcept*: es el identificador del concepto.
- *Resource_idResource*: es el identificador del recurso.
- *similarity*: es el valor de la similaridad entre el concepto y el recurso.

SCD: guarda las similaridades de tipo concepto-diccionario.
- *Concept_idConcept*: es el identificador del concepto.
- *Tag_idTag*: es el identificador de la etiqueta perteneciente al diccionario.
- *similarity*: es el valor de la similaridad entre el concepto y la etiqueta del diccionario.

SRR: guarda las similaridades de tipo recurso-recurso.
- *Resource_idResource1*: es un identificador de recurso.
- *Resource_idResource2*: es otro identificador de recurso. Puede ser el mismo que el de *Resource_idResource1*.
- *similarity*: es el valor de la similaridad entre los dos recursos.

3. Implementación.

3.1. Implementación de *FlickrBackup*.

3.1.1. Metodología de Trabajo.

Debido a *FlickrBackup* es una aplicación muy pequeña, se decidió usar la versión ágil del proceso unificado (*Agile Unified Process*) como metodología para su desarrollo. Así se ha trabajado con una metodología más sencilla y directa pero haciendo uso de UML como en el proceso unificado.

3.1.2. Paradigma de Trabajo.

Para desarrollar esta aplicación, como ya se ha comentado con anterioridad, se ha decidido usar el paradigma orientado a objetos, debido a las facilidades que ofrece para construir una aplicación modular a base de componentes reutilizables.

3.1.3. Planificación.

Para la implementación de la aplicación *FlickrBackup* se estableció un mes como tiempo aproximado para conseguir una versión funcional y capaz llevar a cabo las funcionalidades deseadas.

3.1.4. Tecnología.

Para la implementación de *FlickrBackup* se ha elegido el leguaje de programación *Java*[1] debido a diferentes motivos:

- Por se un lenguaje de programación orientado a objetos.
- Por su compatibilidad con sistemas gestores de bases de datos relacionales.
- Por ser gratuito.
- Por la completa *API* que tiene de base y, debido al popular uso de este lenguaje, la multitud *API's* externos que completan su funcionalidad.

Como sistema gestor de bases de datos se ha escogido *MySQL*[2] por estar basado en lenguaje *SQL*, por ser gratuito y fiable. Además consta de una interfaz gráfica que se puede instalar de forma complementaria y gratuita que te facilita el uso y administración de las bases de datos.

Para la conectividad con *Flickr* se ha usado *flickrj*[3], una biblioteca desarrollada en *Java* que encapsula el *API* ofrecido por *Flickr* para su uso más fácil por aplicaciones desarrolladas en *Java*.

3.1.5. Implementación de la Interfaz de Usuario.

Esta aplicación consta de una interfaz de usuario muy sencilla y basa en texto. Básicamente, tras comenzar la ejecución de la aplicación, hace dos consultas al usuario vía línea de comandos. La aplicación pide que el usuario escriba las etiquetas por las que buscar imágenes en *Flickr* y, si así lo desea, que establezca un límite máximo de imágenes a descargar en la ejecución.

3.2. Implementación de *ACoAR*.

3.2.1. Metodología de Trabajo.

La metodología usada para el desarrollo de *ACoAR* es la del proceso unificado (RUP). Como ya se ha comentado anteriormente, su desarrollo está basado en el trabajo de la tesis doctoral de Francisco Echarte [4], por tanto, la parte de la implantación de *ACoAR* de este proyecto puede considerarse una iteración más dentro del ciclo de vida del proceso unificado, orientada a adaptar el diseño de la aplicación a las condiciones específicas del proyecto y a optimizarlo para el entorno de ejecución específico usado en las pruebas.

3.2.2. Paradigma de Trabajo.

[1] http://www.oracle.com/technetwork/java/index.html
[2] http://www.mysql.com/
[3] http://flickrj.sourceforge.net/

Para desarrollar esta aplicación, al igual que *FlickrBackup*, se ha decidido usar el paradigma orientado a objetos. Esto es debido a que dicho paradigma está basado en la construcción de aplicaciones modulares permitiendo:

1. La fácil implementación de los componentes de los que está compuesto *ACoAR*.
2. La posibilidad de reutilizar ciertos módulos de la aplicación *FlickrBackup*.

Además, haciendo uso de la herencia, la abstracción de datos y el polimorfismo, característicos de este paradigma, facilita el intercambio de métodos, fórmulas y algoritmos en cada uno de los componentes de *ACoAR*.

3.2.3. Planificación.

Originalmente la implementación de esta aplicación estaba planificada para ser acabada en 4 o 5 meses pero, debido a los problemas que surgieron con el consumo de memoria, la incorporación de *memcached* al sistema, los problemas de tiempos de ejecución y las modificaciones del código a consecuencia de ello, este tiempo se alargó a los 7 meses.

A continuación se detalla el tiempo empleado en cada una de las etapas de la implementación de la aplicación *ACoAR*:

Etapa de Desarrollo	Tiempo
Desarrollo del Modelo de Datos	1 mes
Desarrollo de Componentes de *ACoAR*.	1 mes
Desarrollo de Clase principal	1 mes
Instalación de *Membase* e implementación de clases asociadas.	2 meses
Modificaciones de código para optimización de tiempo	2 meses

3.2.4. Tecnología.

Para la implementación de *ACoAR* se ha decidido usar el lenguaje de programación *Java* por diversas razones:

- Por ser un lenguaje de programación orientado a objetos.
- Por su compatibilidad con sistemas gestores de bases de datos relacionales.
- Por ser multiplataforma, ya que esto mejora la portabilidad de la aplicación.
- Por se un lenguaje ampliamente utilizado, lo que posibilita que haya *API's* ya desarrollados para interactuar con otros sistemas, u para ampliar su funcionalidad.
- Por se gratuito.

Para implementar las bases de bases de datos se optó por usar *MySQL*, ya que es un sistema gestor de bases de datos relacionales basado en *SQL*. Además, en la propia página de *MySQL* te proporcionan el *driver* ya escrito en *Java*.

Durante la implementación de *ACoAR* surgieron problemas con la gestión de la memoria relacionados con los vectores de representación y, sobretodo, con las medidas de

similaridad. El problema en sí era que se generaban tantos datos que se llegaba a desbordar la memoria asignada al proceso de java y, por tanto, se detenía la ejecución del programa. Por ello se decidió hacer uso de *Memcached*, un sistema distribuido de memoria caché. En un principio lo que se quería con ello era trasladar los datos correspondientes a los vectores y las similaridades de la memoria del proceso a la proporcionada por *Memcached*. Pero surgieron una serie de problemas.

Problemas con Memcached

Como ya se ha mencionado con anterioridad la aplicación genera tanta información que desborda la memoria asignada al proceso de *Java*. Por ello decidió utilizar un servidor de cacheo, *Memcached*. Concretamente se ha utilizado una distribución del mismo llamada *Membase*[1] que está mejor preparada para correr sobre *Windows*. Con esto, en un principio, quiso aprovechar el resto de la memoria de la máquina (la no ocupada por el proceso de java u los de otras aplicaciones), o incluso las de otras.

Antes de la incorporación de dicho servidor de caché a la aplicación *ACoAR* se decidió hacer una serie de pruebas con *Memcached*. Durante el uso de esta aplicación surgieron diferentes problemas en función de la topología adoptada por el servidor.

1. Se ha instaló el servidor de *Membase*, configurado como *Memcached*, en la misma máquina en la que se ejecutaba el proceso de java. A la hora de ejecutar se encontraron dos tipos de problemas:

    a. Había picos de memoria en el proceso de java. La aplicación, de tener un uso de memoria medio de entre 2 y 5 MB, pasaba a tener picos que superaban los 250 MB llegando a desbordar e interrumpir la ejecución del programa. Como solución a esto se optó por, cada cierto tiempo, poner a dormir el proceso 5 segundos y llamar al recolector de basura de la máquina virtual para que liberase de la memoria aquellos objetos que no estaban en uso. Además también se puso la condición de que si la memoria utilizada superaba los 150 MB llevase a cabo las mismas acciones. Con estas medidas el problema del pico de memoria, más o menos, andaba controlado.
    b. Al insertar gran número de valores en Memcached, decenas de millones, al intentar recuperar los más antiguos se comprobó que estos ya no estaban almacenados en caché. Se observó, que esto era debido a que llegado cierto punto en que la memoria asignada a *Memcached* estaba casi llena, este comenzaba a sobreescribir los valores antiguos por los nuevos insertados. Se creyó que esto era debido a que la máquina no tenía suficiente memoria, así que como solución, se optó por realizar cambios en la topología de *Memcached* para así conseguir un aumento de la memoria disponible.

2. Otra de las topologías probadas fue instalar *Membase* en la misma máquina en que se ejecuta la aplicación y en una máquina dedicada, comunicándolas entre sí creando un cluster. De esta manera, al ejecutar las pruebas, saltaban muchas excepciones y mensajes de alerta del servidor relacionados con descartes de operaciones, servidor ocupado y fallos de conexión. Además, la memoria volvía a

[1] http://www.couchbase.org/membase

dar importantes picos pero a mayor frecuencia y superiores en intensidad. Además al intentar recuperar los valores supuestamente almacenados en el servidor, encontramos que no había llegado a almacenar nada. Debido a esto, se optó por ir paso a paso y se simplificó la topología de *Membase* para ver si era esto la causa de los problemas.

3. La siguiente topología puesta a prueba fue instalar *Membase* en una sola máquina dedicada. Al realizar las pruebas vimos que surgían los mismos problemas que en la topología anterior. Por tanto, se dedujo que estarían relacionados con el hecho de la comunicación en red de las máquinas. Esto podría ser por el hecho de que la aplicación de prueba generase más mensajes al servidor de los que éste y la red pudiesen procesar. Siguiendo esta idea se intentó buscar soluciones.

    a. Lo primero que se probó fue lo mismo que en la primera topología, realizar cada cierto tiempo pausas en la aplicación y llamar al recolector de basura de la máquina virtual. Se observó que a mayor frecuencia y duración de dichas paradas, menos errores se daban, pero, eso sí, la tasa de inserciones en la memoria caché era más baja y, por tanto, la ejecución de la aplicación más lenta.
    b. Debido a que esto no era una solución completa, ni muy elegante, se decidió seguir buscando alternativas. Revisando la documentación del API usado como interfaz con *Memcached* se observó que al realizar una inserción de un objeto en la caché el método en cuestión devolvía un objeto futuro (*Future*). Esta clase representa el resultado de una operación asíncrona, en este caso, el almacenamiento del objeto en la memoria caché. Entonces, esperando a que esta clase reciba un *booleano* que indique que se ha insertado el objeto, nos aseguramos que *Memcached* ha recibido, llevado a cabo la acción y, por tanto, la ha eliminado de la cola de entrada. Al realizar este cambio en el cliente lo primero observado fue que ya no salían las excepciones y alertas asociadas a descartes de operaciones, servidor ocupado y fallos de conexión, pero con la contrapartida de que la aplicación se había vuelto más lenta, debido a que ésta tenía que esperar respuesta del servidor.

Tras conseguir la estabilidad en las pruebas con *Membase* se siguió testeando la memoria necesaria para albergar las diferentes tablas de vectores y similaridades. Debido a que incluso con 2 servidores de *Membase*, haciendo un total de 3GB de RAM, no era suficiente para contener dichas tablas se optó por la solución de utilizar la base de datos como almacén persistente durante la ejecución y usando *Membase* como capa intermedia entre la BBDD y el proceso de java para ahorrar tiempo de ejecución minimizando lecturas a disco. Esta solución vuelve más lenta la aplicación debido a que es necesario escribir todas las modificaciones en disco en vez de tenerlas en memoria, pero gracias a *Membase* podemos almacenar parte de los valores en memoria y no es necesario hacer todas las lecturas en disco.

3.2.5. Cambios de Diseño.

Durante la implementación de *ACoAR*, debido a diferentes problemas, se hicieron algunos cambios en el diseño en algunos de los componentes de la aplicación.

Durante gran parte de la implantación de *ACoAR*, el cálculo de similaridades recurso-recurso (SRR) se hacia de forma que se comparaban todas las combinaciones de recursos. Viendo que, debido a esto, la ejecución de *ACoAR* duraba alrededor de 3 días, se decidió reducir el número de combinaciones. Esto se consiguió calculando solamente las similaridades entre recursos que pertenezcan al mismo concepto, o entre recursos de conceptos cuya similaridad esté por encima de cierto umbral. Con esto, se consiguió reducir el número de cálculos de similidaridad, dejando aquellos con la probabilidad de que sea más alta.

También hubo un cambio en el criterio de convergencia. Originalmente se compara el número de anotaciones del recurso en cuestión con un cierto umbral. Esto implicaba hacer el cálculo de número de anotaciones del recurso, que era una operación relativamente costosa en comparación con la nueva propuesta. El método actual implementado como criterio de convergencia se basa en comparar el número de anotaciones en el recurso de etiquetas pertenecientes al diccionario. Con esto se ahorra tiempo de ejecución debido a que para calcular ese número de anotaciones solamente hay que sumar las componentes del vector representativo del recurso.

3.2.6. Implementación de la Interfaz de Usuario.

Esta implementación de *ACoAR* carece completamente de interfaz de usuario.

4. Resultados Experimentales.

4.1. Resultados Experimentales de *FlickrBackup*.

4.1.1. Escenario de Trabajo.

Para la descarga de imágenes médicas mediante la aplicación *FlickrBackup* se dispuso de un único ordenador con conexión a *Internet*, donde se instaló la aplicación y *MySQL* con una instancia de la base de datos de imágenes médicas.

4.1.2. Pruebas experimentales.

Una vez construida la aplicación *FlickrBackup* pudimos empezar a descargar imágenes médicas de *Flickr*. Este sitio puede considerarse como un almacén de imágenes de propósito general, en el sentido de que guarda imágenes de todo tipo y temática. Por tanto hay que ir cribando las imágenes para quedarnos solamente con aquellas que nos interesan.

Para realizar este filtrado y quedarnos solamente con las imágenes médicas seguimos un procedimiento que constaba de tres pasos:

1. Poníamos a ejecutar *FlickrBackup* con una serie de etiquetas que considerábamos que era factible que estuvieran anotadas en imágenes médicas.
2. Una vez se hubiesen descargado las imágenes, se repasaban una a una de forma manual, a falta de un método mejor. La que fuese imagen médica se quedaba en el sistema, la que no era borrada. Esto tenía que hacerse de forma manual debido a que es muy difícil automatizarlo para que un programa distinga si una imagen es médica o no basándose sólo en el contenido de la imagen.

3. Tras esto, volvíamos al paso 1 con una nueva combinación de etiquetas.

Seguimos este procedimiento hasta que no encontramos ya ninguna combinación de etiquetas, que no hubiésemos probado, que nos diese un porcentaje aceptable de imágenes médicas entre sus resultados.

4.1.3. Análisis de los Resultados.

Como ya era esperado, el mayor número de imágenes que se descargaron eran imágenes no médicas, y por tanto no nos eran útiles para el desarrollo del proyecto. En total se descargaron 29.163 imágenes, de las cuales solamente resultaron útiles 8.579 de ellas. La figura 3.9 es una representación gráfica de las proporciones de los tipos de imágenes obtenidos.

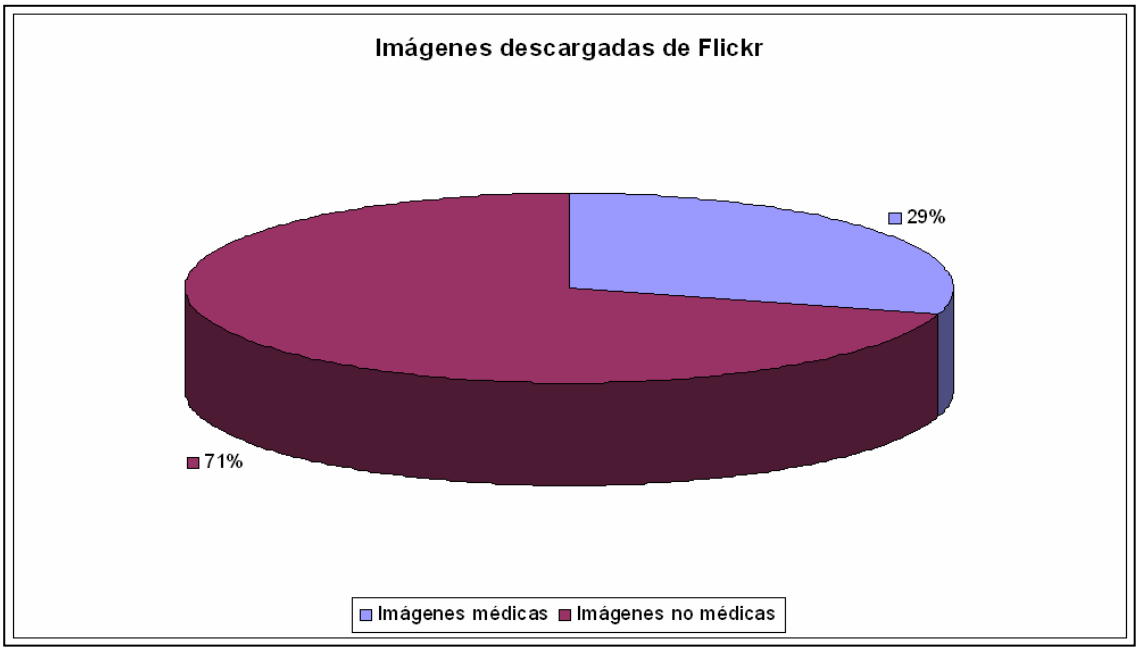


Figura 4.1. Gráfico Circular de los porcentajes de los tipos de imágenes descargadas de Flickr.

Como puede observarse en el gráfico la gran mayoría de las imágenes, el 71%, tuvieron que ser desechadas, frente al 29% que han podido aprovecharse para el proyecto.

Con las 8.579 imágenes que nos quedamos también almacenamos toda una serie de información de las mismas. Con ellas se obtuvo:

- 5.478 Etiquetas.
- 58.878 Anotaciones (apariciones de las etiquetas en las imágenes).
- 1.988 Usuarios.
- 1.270 Notas.

4.1.4. Conclusiones.

Debido a que *Flickr* almacena imágenes de cualquier tipo y no está especializado en imágenes médicas, no es extraño el haber encontrado que la gran mayoría de fotografías descargadas no son imágenes médicas, aun habiendo acotado a imágenes con etiquetas con términos médicos.

Una vez finalizado este proceso hemos conseguido obtener una folksonomía de imágenes médicas con las que poder trabajar. Por tanto ya nos es posible pasar a la siguiente fase del proyecto.

4.2. Resultados Experimentales de *ACoAR*.

4.2.1. Escenario de Trabajo.

Para realizar las pruebas experimentales de la aplicación *ACoAR* se ha dispuesto de dos ordenadores, conectados en red, dispuestos de la siguiente forma:

- Ordenador 1:
  - *MySQL* con la base de datos de imágenes médicas y con una instancia de la base de datos de *ACoAR*.
  - La aplicación *ACoAR*.
  - *Membase* instalado y configurado como *Memcached*.
- Ordenador 2:
  - *MySQL* con una instancia de la base de datos de *ACoAR*.
  - La aplicación *ACoAR*.
  - *Membase* instalado y configurado como *Memcached*.

Cada uno de los ordenadores ejecuta pruebas de forma independiente, salvo el ordenador 2 que obtiene los datos de la folksonomía de imágenes médicas del ordenador 1.

4.2.2. Pruebas experimentales.

Para validar la implementación de *ACoAR* desarrollada en este proyecto se ha decido hacer un conjunto de pruebas. Cada una de estas pruebas va a tener una configuración diferente de ciertos parámetros de la aplicación. El total de parámetros configurables son:

- *dictionaryMina*: Pertenece a la clase *DictionaryNAnnotations*. Representa el número mínimo de anotaciones que debe tener una etiqueta para pasar a formar parte del diccionario. Define qué etiquetas pertenecen al diccionario u cuales no. Toma valores entre 1 y el máximo de anotaciones que toma una etiqueta en la folksonomía.
- *convergenceMina*: Pertenece a la clase *ConvergenceNDictionary*. Indica el número mínimo de anotaciones de las etiquetas del diccionario que tiene que tener un recurso para converger. Puede tomar valores entre 1 y el máximo de anotaciones que tenga un recurso en la folksonomía.

- *mergingThreshold*: Pertenece a la clase *MergingSimilar*. Indica el valor mínimo que tiene que tomar la similaridad entre dos conceptos para unirlos. Puede tomar valores entre 0 y 1.
- *classifierThreshold*: Pertenece o a *ClassifierDelta*, o a *ClassifierSim*. Indica el valor mínimo de delta o de la similaridad entre recurso y concepto respectivamente, para clasificar un recurso en cierto concepto. Puede tomar valores entre 0 y 1.
- *similaritiesMinv*: Pertenece a la clase *Similarites*. Representa el valor mínimo que debe tomar una similaridad para ser almacenada en la base de datos. Puede tomar valores entre 0 y 1.
- *similaritiesSRRThres*: Pertenece a la clase *Similarities*. Indica el valor mínimo que debe tomar la similaridad entre dos conceptos para calcular las similaridades entre los recursos pertenecientes a cada uno. Debe tomar valores entre 0 y el valor de *mergingThreshold*.
- *classifierT*: Indica que clasificador va a usarse. Puede tomar los valores "*sim*" y "*delta*". Cada uno de ellos instancia o *ClassifierSim* o *ClassifierDelta* respectivamente.

A continuación, la figura 4.2 detalla las diferentes combinaciones de valores de los parámetros para las diferentes pruebas.

ID Prueba	*dictionaryMina*	*convergenceMina*	*mergingThreshold*	*classifierThreshold*	*similaritiesMinv*	*similaritiesSRRThres*	*classifierT*
1	100	1	0,5	0,1	0,1	0,4	delta
2	500	1	0,5	0,1	0,1	0,4	delta
3	2564	1	0,5	0,1	0,1	0,4	delta
4	100	1	0,5	0,1	0,1	0,4	sim
5	500	1	0,5	0,1	0,1	0,4	sim
6	2564	1	0,5	0,1	0,1	0,4	sim
7	500	1	0,75	0,1	0,1	0,4	delta
8	500	1	0,9	0,1	0,1	0,4	delta
9	500	1	0,5	0,3	0,1	0,4	delta
10	500	1	0,5	0,5	0,1	0,4	delta
11	500	1	0,5	0,1	0,2	0,4	delta
12	500	1	0,5	0,1	0,5	0,4	delta
13	100	1	0,75	0,1	0,1	0,4	delta
14	2564	1	0,75	0,1	0,1	0,4	delta
15	100	1	0,9	0,1	0,1	0,4	delta
16	100	1	0,5	0,3	0,1	0,4	delta
17	100	1	0,5	0,5	0,1	0,4	delta
18	100	1	0,5	0,1	0,2	0,4	delta
19	100	1	0,5	0,1	0,5	0,4	delta

Figura 4.2. Pruebas de ACoAR.

4.2.2.1. Pruebas.

A continuación va a hacerse una exposición detallada de cada prueba. Se va a mostrar gráficas con los tiempos de ejecución de cada componente de *ACoAR*, gráficas con el número de recursos por concepto, y, además, el número de recursos clasificados, los no clasificados, y dentro de estos últimos, aquellos que han quedado convergidos y los que no.

4.2.2.1.1. Prueba 1.

La configuración de los parámetros de la aplicación para esta prueba fue la siguiente:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

La figura 4.3 muestra la gráfica de los tiempos de ejecución de los componentes de *ACoAR* para la *prueba 1*. Como puede observarse la mayor parte del tiempo de ejecución está dedicado al cálculo de la similaridades recurso-recurso, siendo prácticamente insignificante el tiempo dedicado a otras tareas en comparación con ésta.

La figura 4.4 muestra un grafica donde se presenta el número de recursos que tiene cada concepto generado durante la *prueba 1*. Vemos que destacan, por su población, los cuatro primeros conceptos presentados, siendo aquel con nombre *ultrasound* el más poblado por más del doble que los demás. Lo más probable que esto sea debido a que en la folksonomía utilizada como datos de entrada había un gran número de ecografías.

Más datos que pueden darse de esta prueba son:

- Nº de Conceptos: 21.
- Nº de Recursos clasificados: 5953.
- Nº de Recursos no Clasificados: 2626.
- Nº de Recursos *Pending*: 173.
- Nº de Recursos *Converged*: 2453.

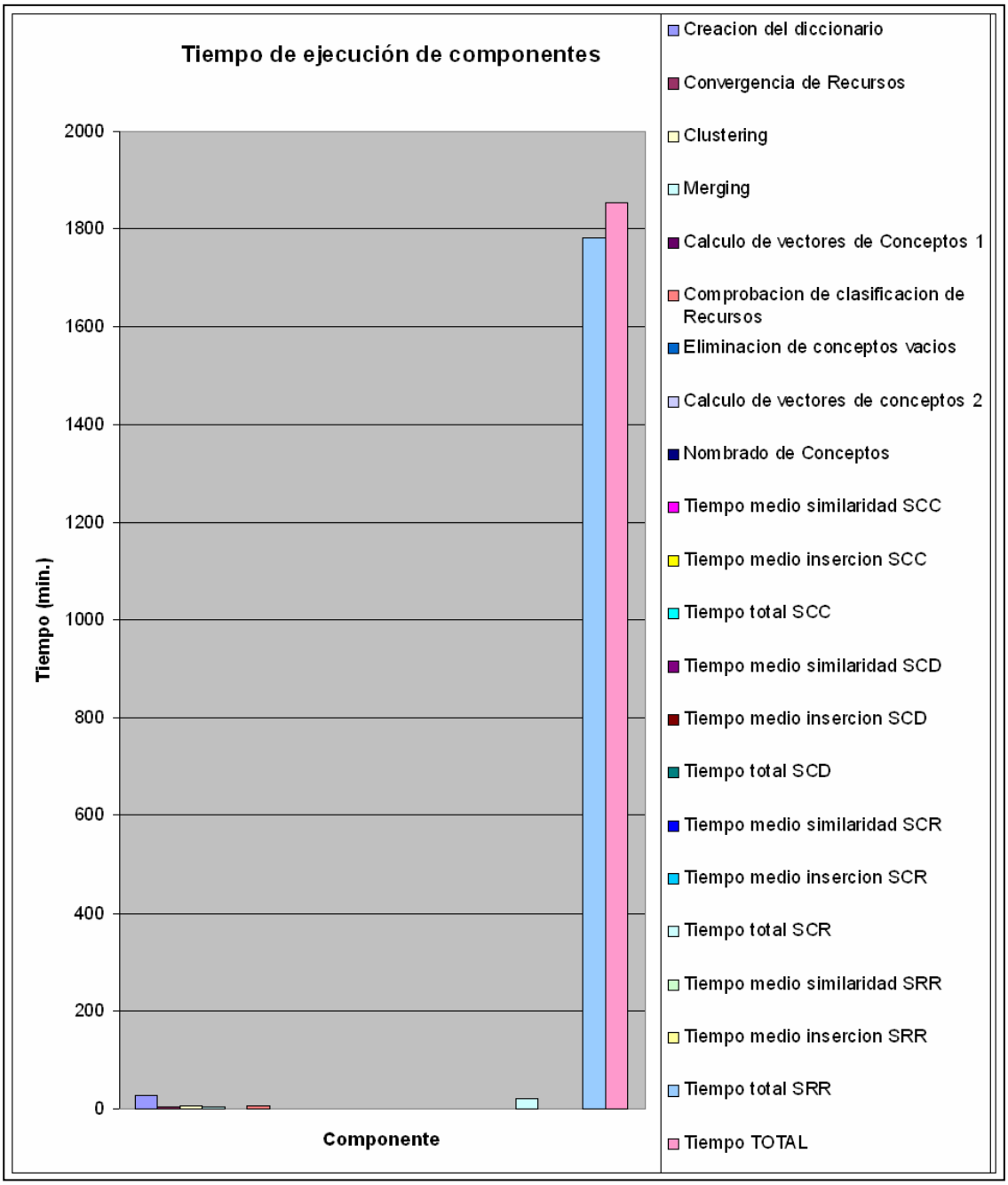


Figura 4.3. Tiempo de Ejecución de Componentes de la Prueba 1.

Como puede observarse en la figura 4.5 y los datos dados con anterioridad son mayoría los recursos clasificados en algún concepto, siendo éstos alrededor del 70%.

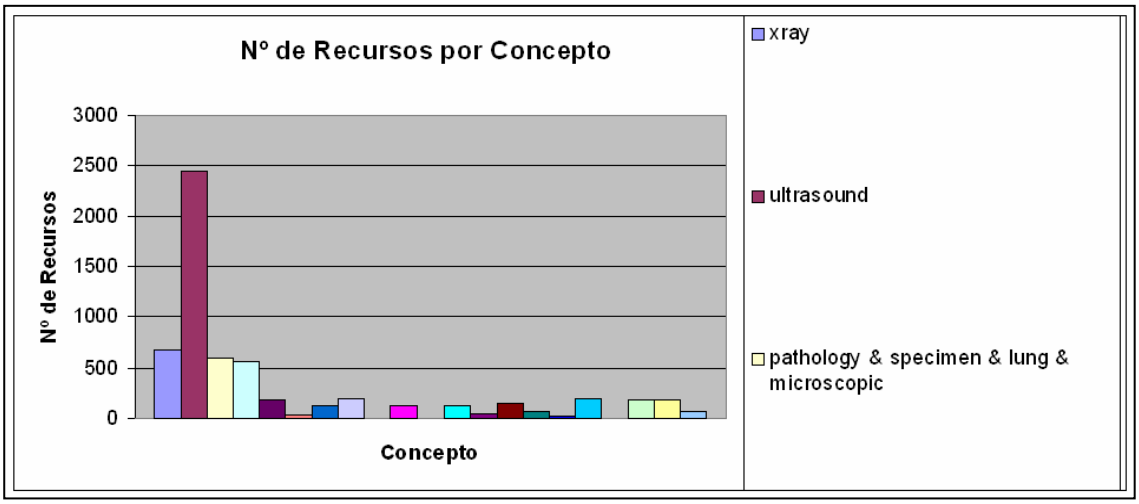


Figura 4.4. Número de Recursos por Concepto de la Prueba 1.

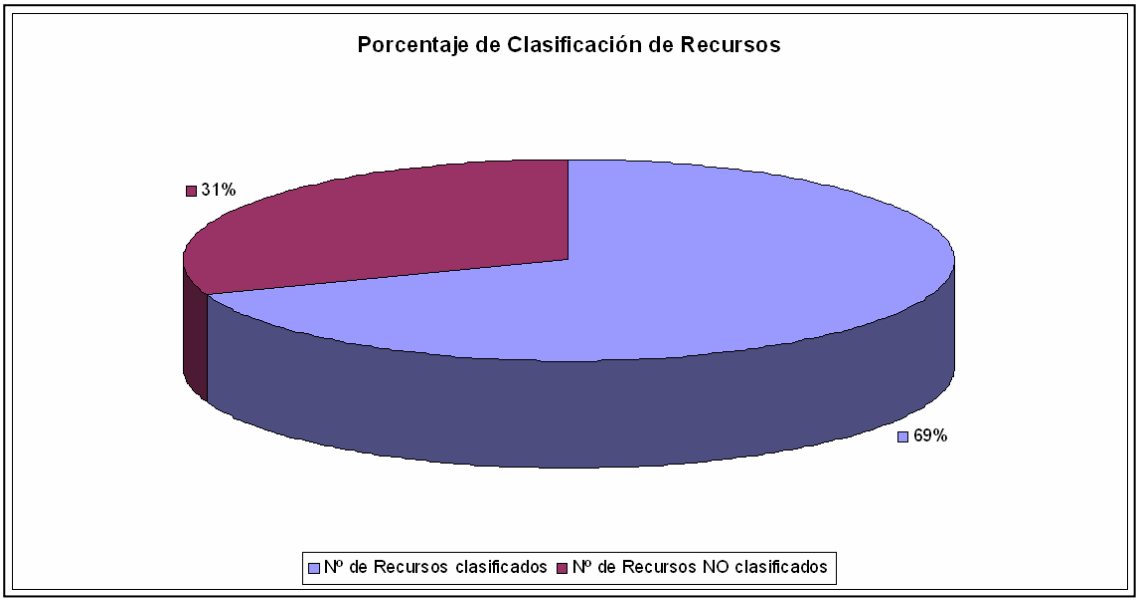


Figura 4.5. Porcentaje de clasificación de Recursos de la Prueba 1.

4.2.2.1.2. Prueba 2.

La configuración de los parámetros de la aplicación para esta prueba fue la siguiente:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

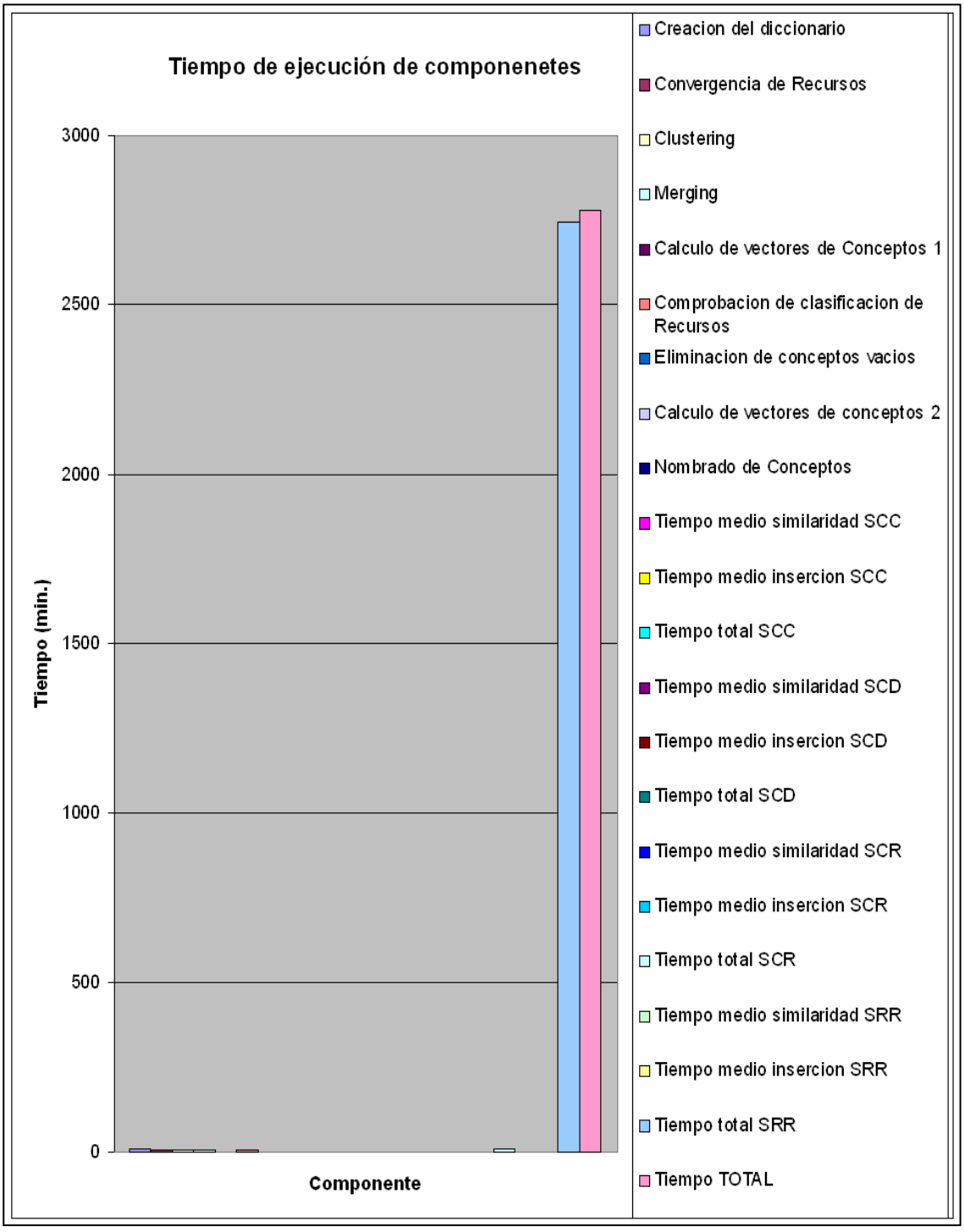


Figura 4.6. Tiempo de Ejecución de Componentes de la Prueba 2.

La figura 4.6 contiene los tiempos de ejecución de los componentes de *ACoAR* durante la prueba. Igual que en la prueba anterior, la gran mayoría del tiempo de ejecución empleado es usado para el cálculo de similaridades relación-relación. El tiempo empleado es mayor que la prueba anterior ya que ha tenido que comparar más recursos, debido al menor número de conceptos.

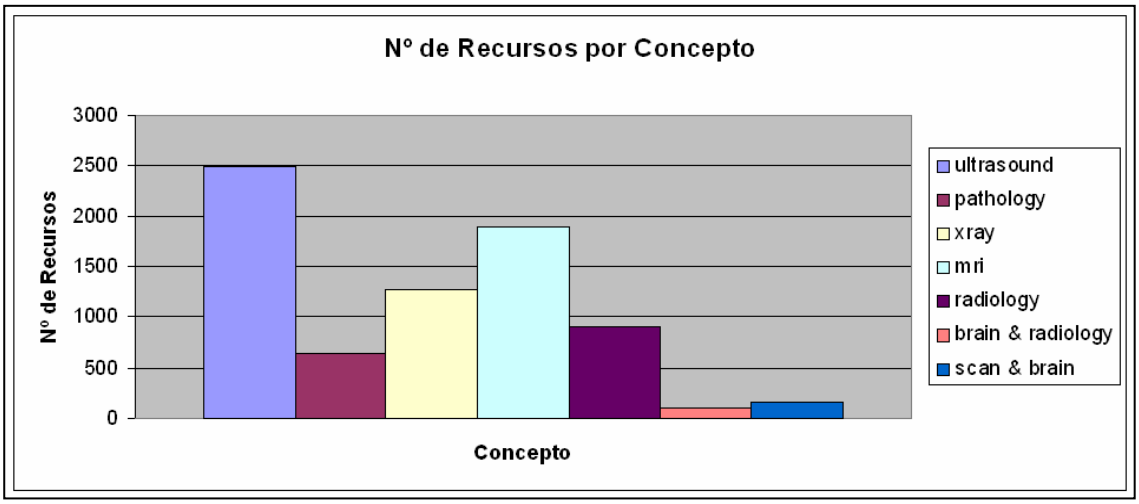


Figura 4.7. Número de Recursos por Concepto de la Prueba 2.

En esta prueba se han creado menos conceptos que en la anterior. Esto es debido a que hay menos etiquetas en el diccionario, ya que en esta prueba se ha endurecido la condición que tiene que cumplir las etiquetas para ello. Aun así, puede verse en la figura 4.7 que sigue siendo el concepto *ultrasound* el más poblado, aunque con menor diferencia con los demás conceptos ya que el resto de recursos clasificados lo ha hecho en un número menor de conceptos.

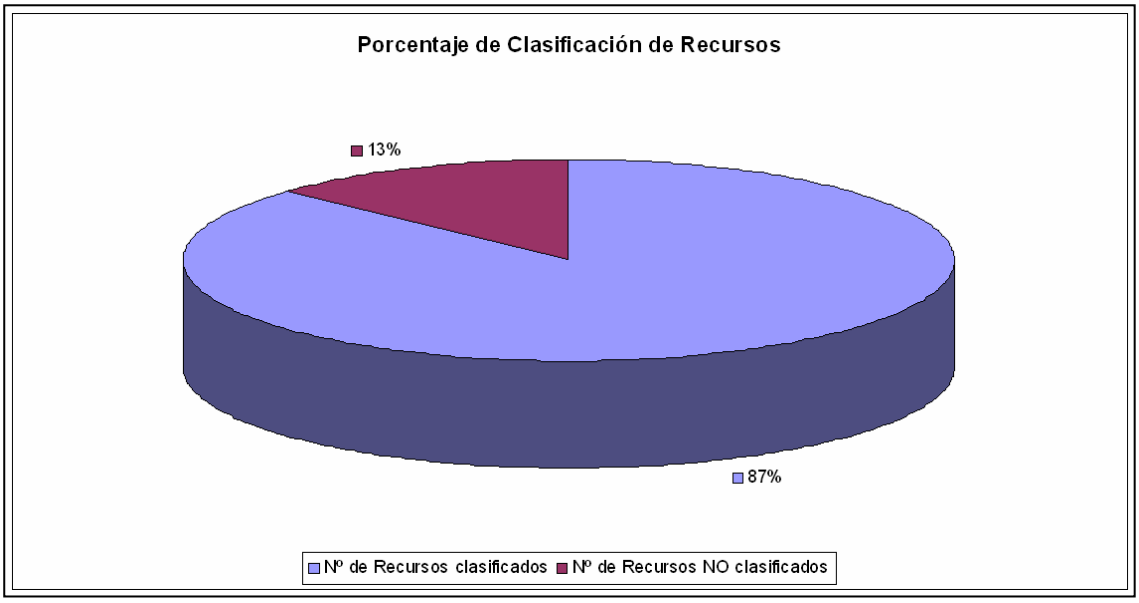


Figura 4.8. Porcentaje de Clasificación de Recursos de la Prueba 2.

La figura 4.8 muestra que la clasificación ha subido del 70% al 87% de los recursos totales del sistema.

Otros datos de la prueba son:

- Nº de Conceptos: 7.
- Nº de Recursos clasificados: 7470.
- Nº de Recursos no Clasificados: 1109.

- Nº de Recursos *Pending*: 25.
- Nº de Recursos *Converged*: 1084.

4.2.2.1.3. Prueba 3.

La configuración de los parámetros de esta prueba fue la siguiente:

- *dictionaryMina*: 2564.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

La figura 4.9 muestra los tiempos de ejecución de los componentes. Otra vez, el mayor tiempo empleado lo ha ocupado el cálculo de similaridades recurso-recurso.

La figura 4.10 es prácticamente trivial. Nos muestra que solamente se ha generado un concepto durante la ejecución y, obviamente, todos los recursos clasificados pertenecen a dicho concepto. También puede verse que este concepto no es otro que el de *ultrasound*.

En la figura 4.11 puede verse que se han invertido los porcentajes, dejando el 70% de los recursos sin clasificar, debido a la falta de conceptos donde clasificarlos.

Otros datos de la prueba son:

- Nº de Conceptos: 1.
- Nº de Recursos clasificados: 2564.
- Nº de Recursos no Clasificados: 6015.
- Nº de Recursos *Pending*: 25.
- Nº de Recursos *Converged*: 5990.

Tras ver los resultados de las tres primeras pruebas podemos sacar una conclusión: que, a mayor valor de *dictionaryMina*, entonces, menos etiquetas en el diccionario y menor número de conceptos.

También podemos ver que parece que no hay una dependencia directa entre el número de conceptos generados y el porcentaje de recursos clasificados.

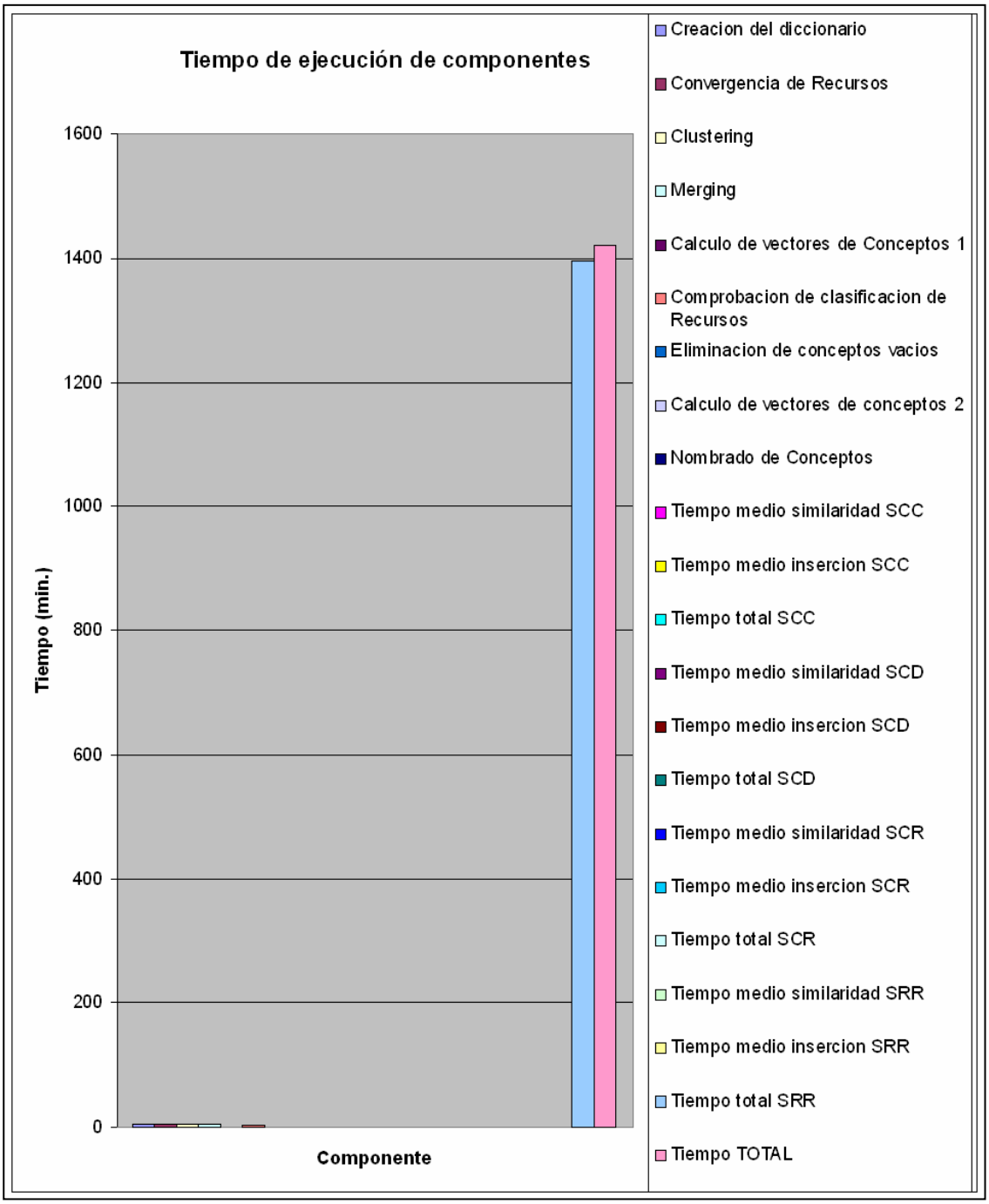


Figura 4.9. Tiempo de Ejecución de Componentes de la Prueba 3.

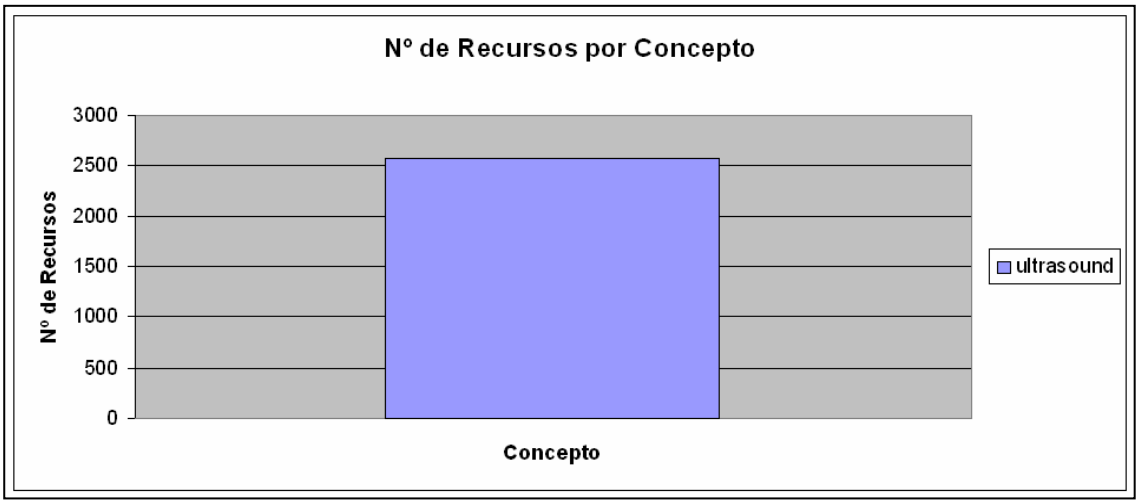


Figura 4.10. Número de Recursos por Concepto de la Prueba 3.

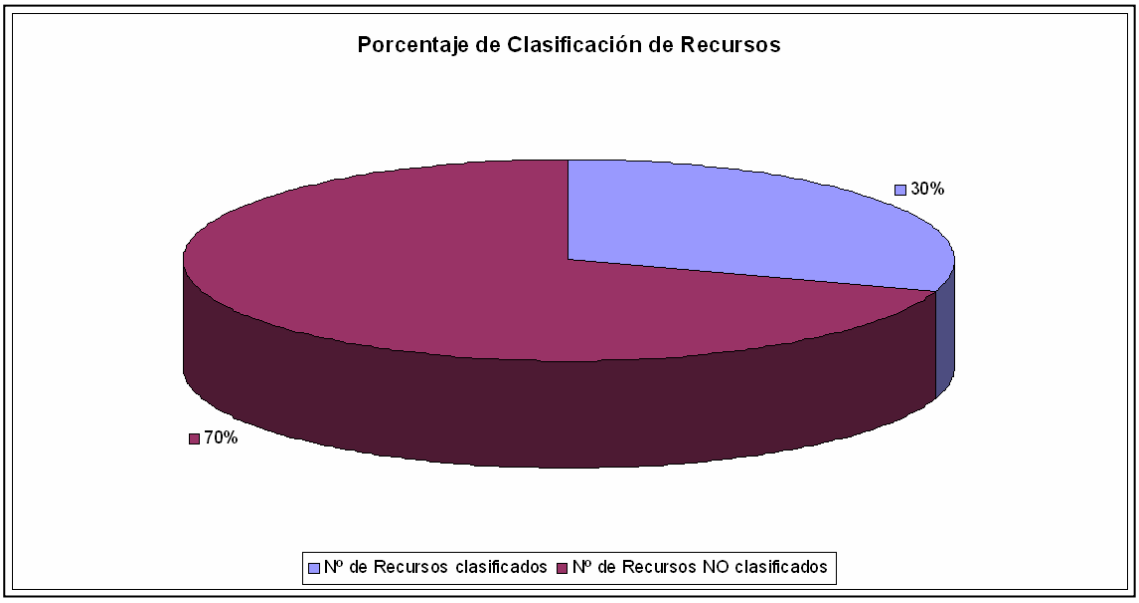


Figura 4.11. Porcentaje de Clasificación de Recursos de la Prueba 3.

4.2.2.1.4. Prueba 4.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim.

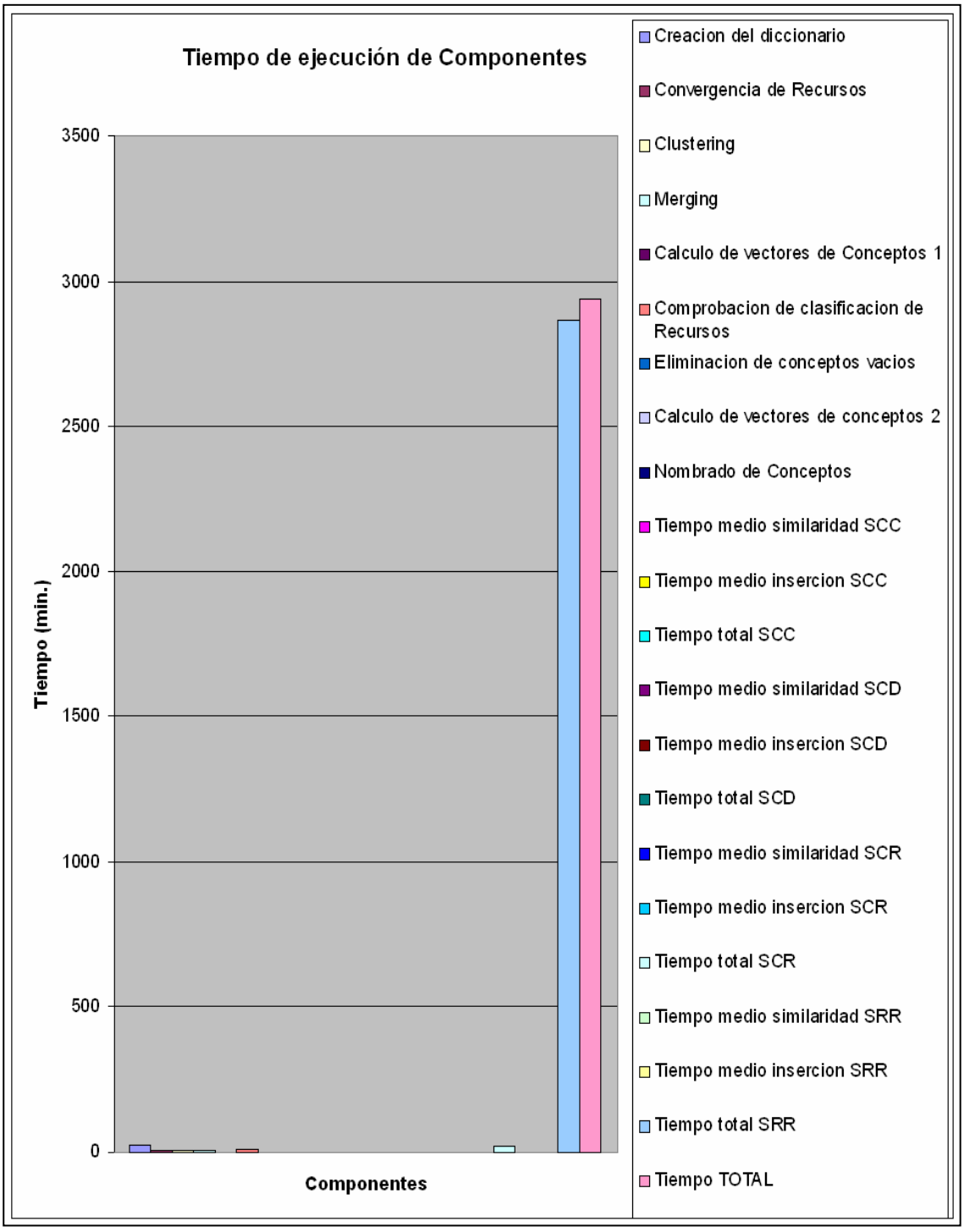


Figura 4.12. Tiempo de Ejecución de Componentes de la Prueba 4.

Vemos también en esta prueba que la mayoría del tiempo de ejecución es empleado para el cálculo de las similaridades recurso-recurso.

En la figura 4.13 puede verse que los conceptos más poblados son *ultasound* y *mri*, siguiéndoles de cerca *xray*.

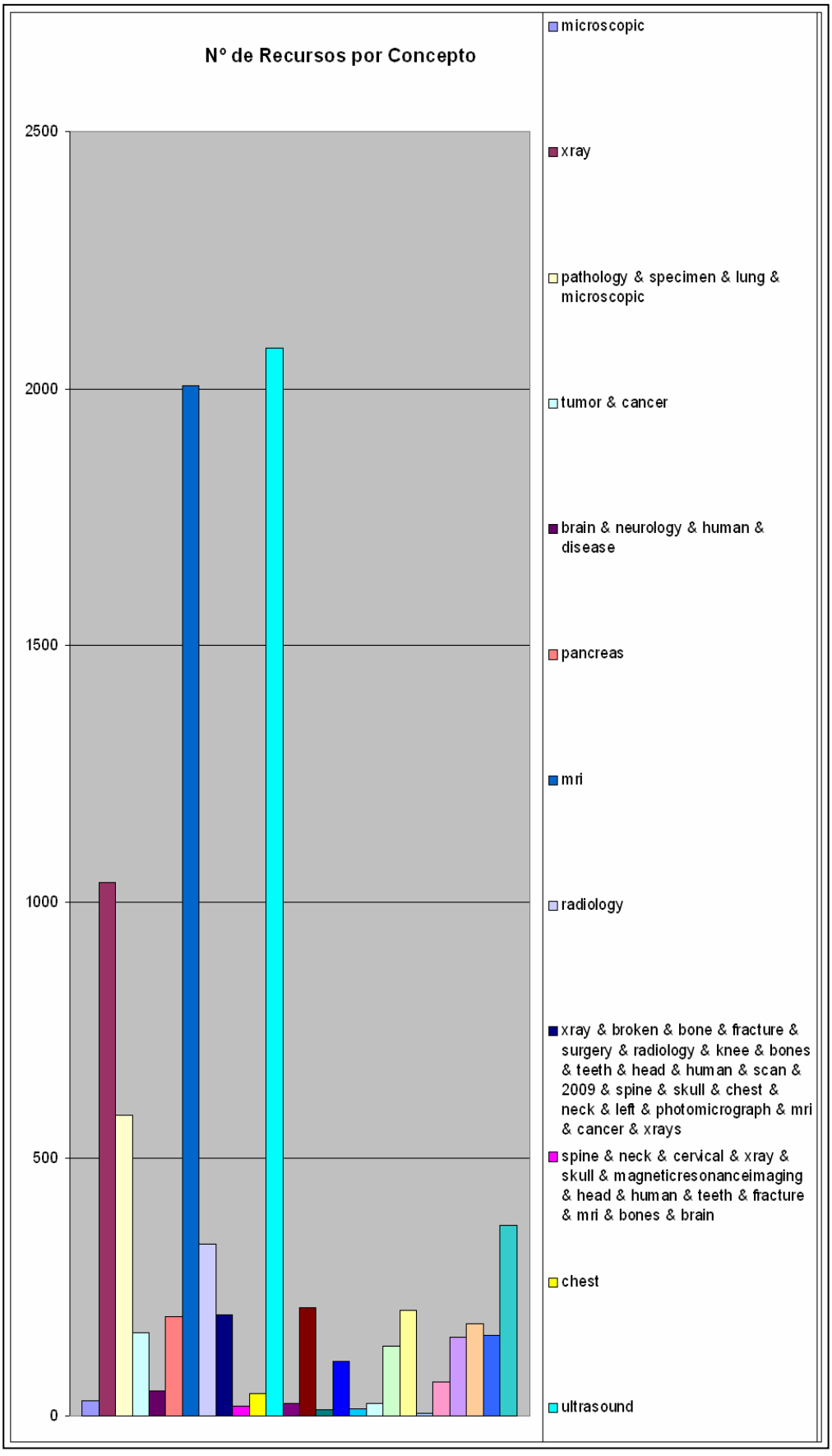


Figura 4.13. Número de Recursos por Concepto.

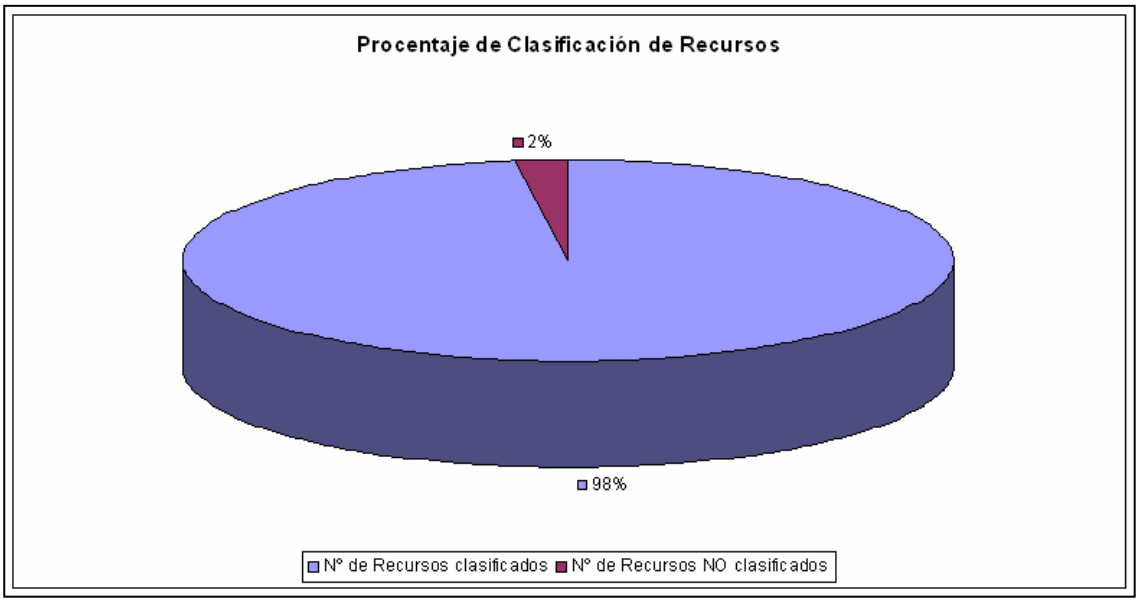


Figura 4.14. Porcentajes de Clasificación de Recursos

En la figura 4.14 puede observarse que, en esta prueba, se ha conseguido clasificar prácticamente a todos los recursos del sistema, el 98% de ellos.

A continuación se detallan otros datos de la prueba:

- Nº de Conceptos: 26.
- Nº de Recursos clasificados: 8397.
- Nº de Recursos no Clasificados: 182.
- Nº de Recursos *Pending*: 173.
- Nº de Recursos *Converged*: 9.

4.2.2.1.5. Prueba 5.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim.

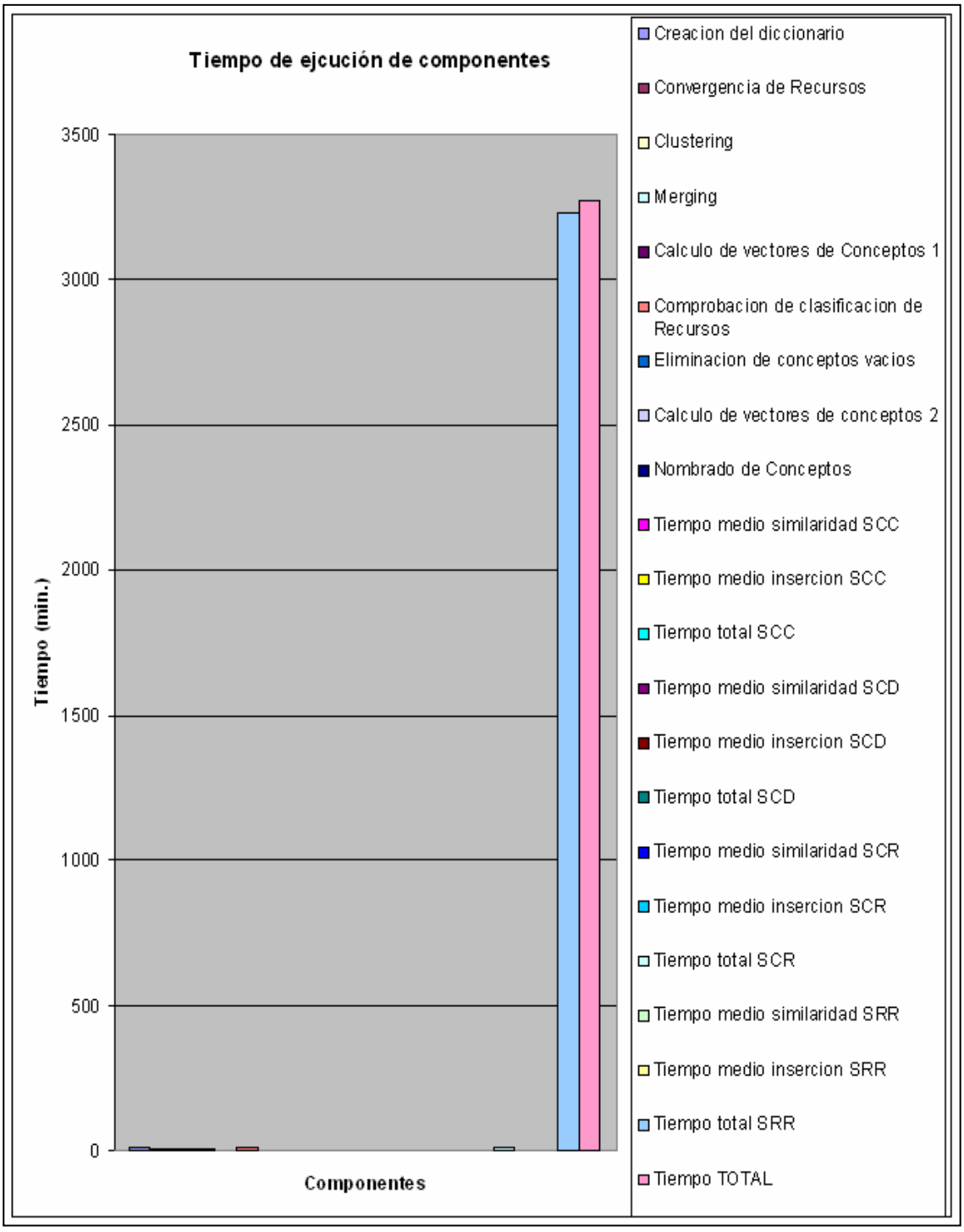


Figura 4.15. Tiempo de Ejecución de Componentes de la Prueba 5.

Otra vez la mayor parte del tiempo de ejecución de la prueba es usado por el módulo de cálculo de similaridades para calcular aquellas de tipo recurso-recurso.

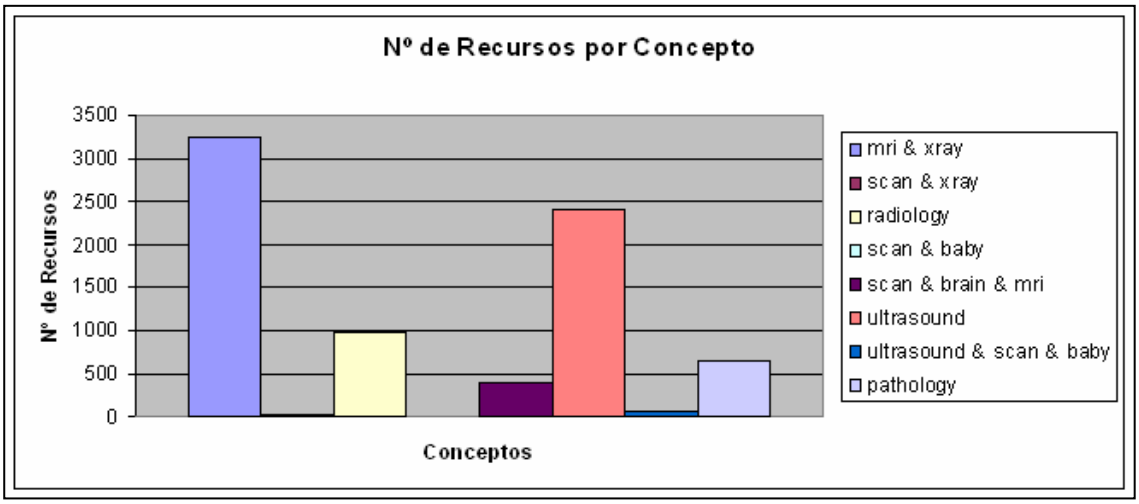


Figura 4.16. Número de Conceptos por Recurso de la Prueba 5.

La figura 4.16 nos muestra que el concepto con mayor población, esta vez, es el de *"mri & xray"* seguido de cerca de *ultrasound*.

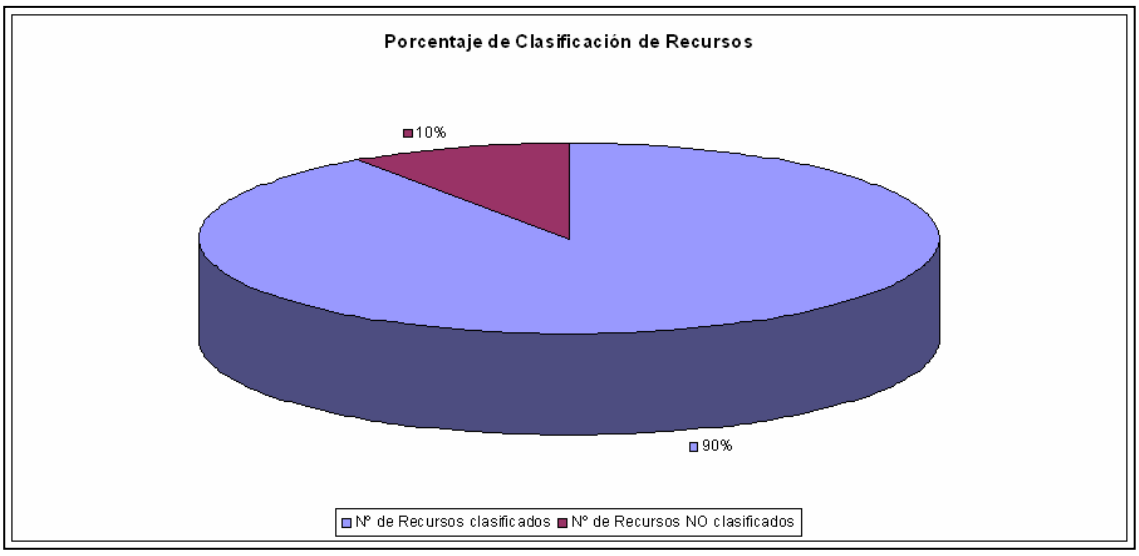


Figura 4.17. Porcentaje de Clasificación de Recursos.

En la figura 4.17 puede verse que en esta prueba también se ha conseguido un buen porcentaje de clasificación de los recursos del sistema, llegando este hasta el 90%.

Los datos correspondientes a la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 8.
- Nº de Recursos clasificados: 7756.
- Nº de Recursos no Clasificados: 823.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 1.

4.2.2.1.6. Prueba 6.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 2564.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim.

En la figura 4.18 puede observarse, otra vez, que el resto de tiempos de ejecución de los componentes queda prácticamente reducido a algo insignificante en comparación por el utilizado para el cálculo de las similaridades recurso-recurso.

La figura 4.19 muestra que solamente se ha creado un concepto. Esto es debido a que sólo hay una etiqueta en el diccionario, *ultasound*. Por tanto solamente puede crearse un concepto y, solamente pueden clasificarse aquellos recursos que tengan anotaciones de dicha etiqueta.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 1.
- Nº de Recursos clasificados: 2564.
- Nº de Recursos no Clasificados: 6015.
- Nº de Recursos *Pending*: 6015.
- Nº de Recursos *Converged*: 0.

En esta prueba puede verse que ha pasado lo mismo, en cuestión de porcentaje de recursos clasificados, que en la prueba 3. Vemos que la mayoría de los recursos han quedado sin clasificar, el 70%. Esto es debido a que solamente hay un concepto donde clasificarlos, definido sólo por una etiqueta, *ultrasound*, y, por tanto, sólo se clasifican aquellos recursos que contengan dicha etiqueta.

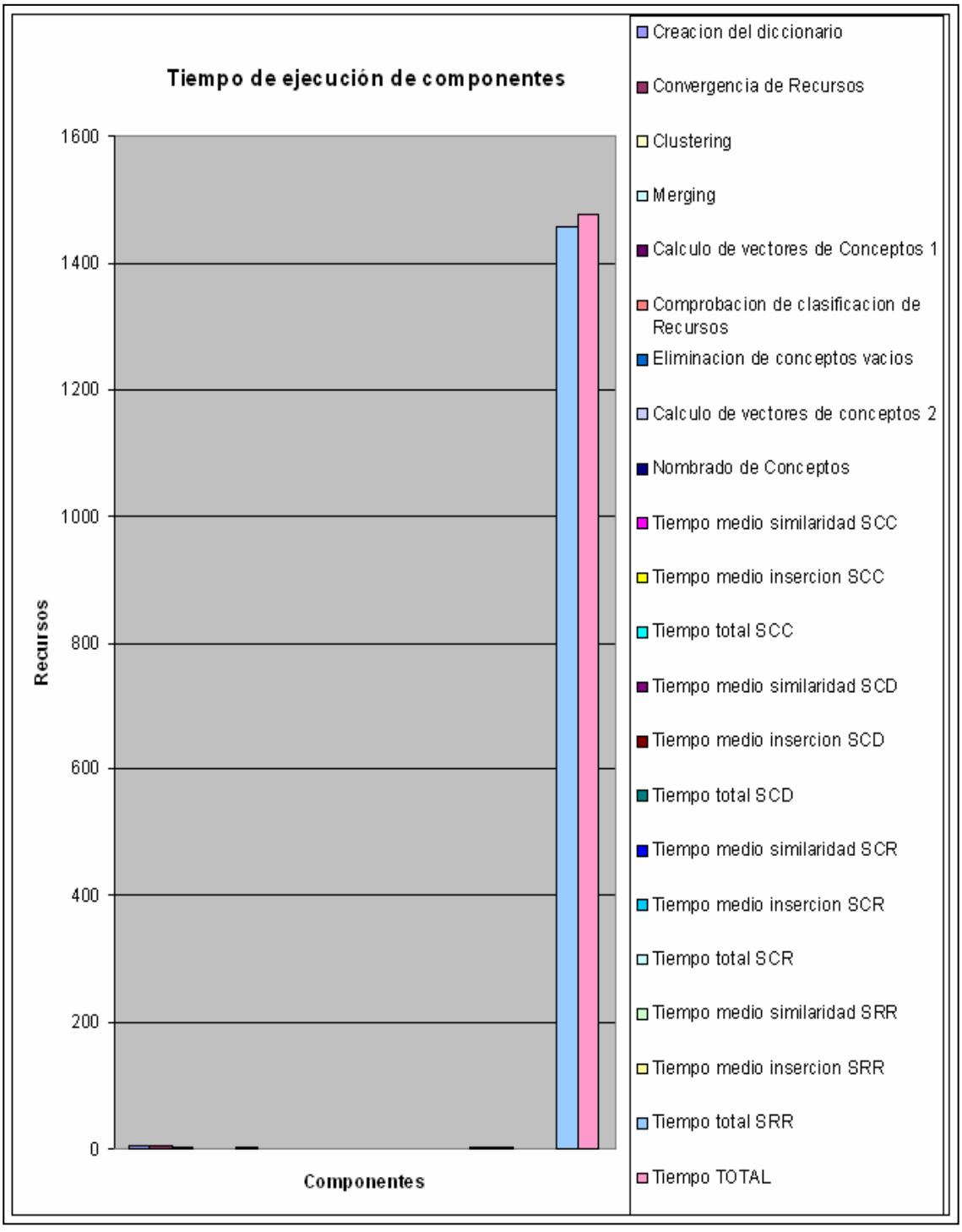


Figura 4.18. Tiempos de Ejecución de Componentes de la Prueba 6.

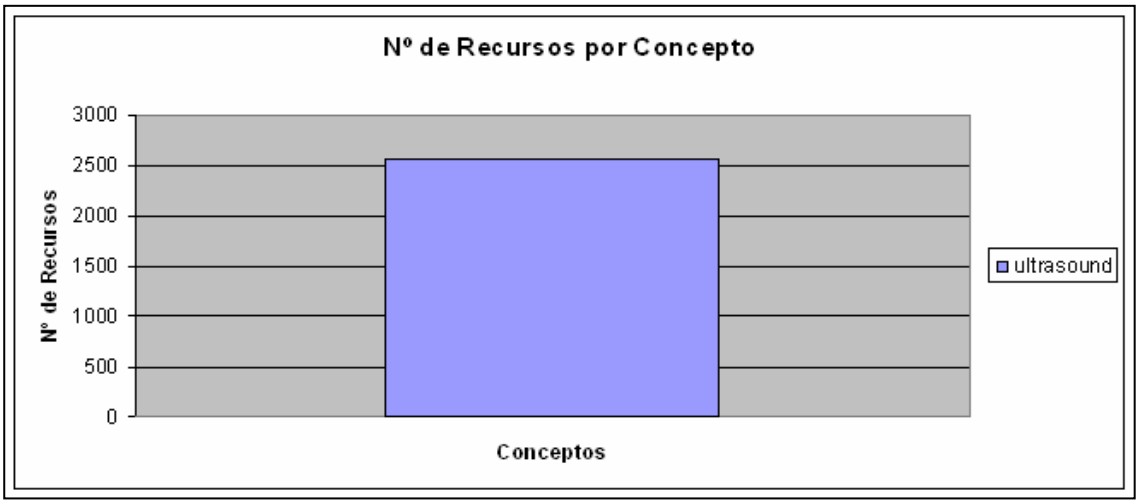


Figura 4.19. Número de Recursos por Concepto de la Prueba 6.

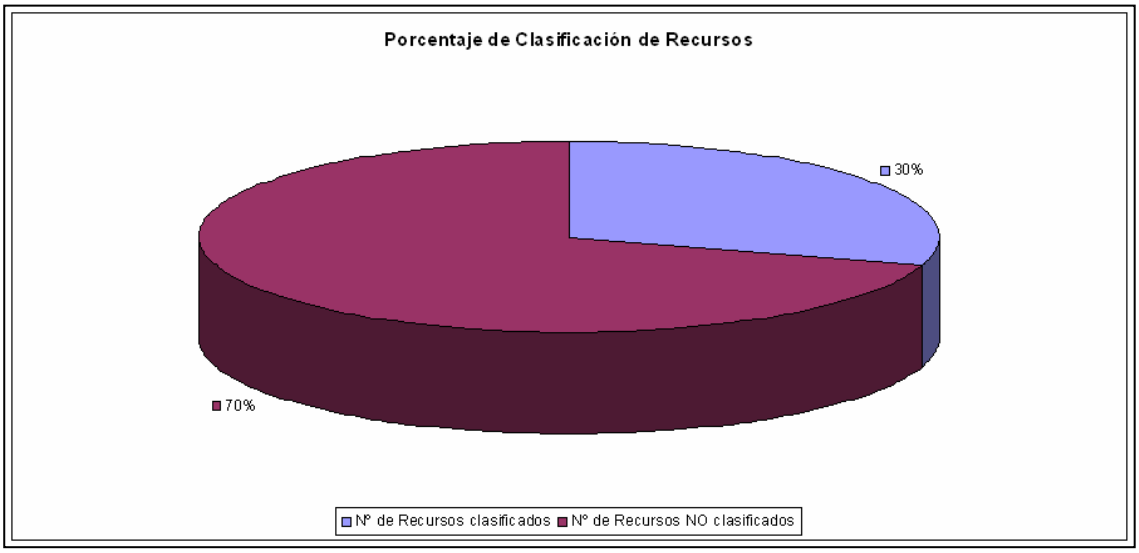


Figura 4.20. Porcentaje de Clasificación de Recursos de la Prueba 6.

4.2.2.1.7. Prueba 7.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.75
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

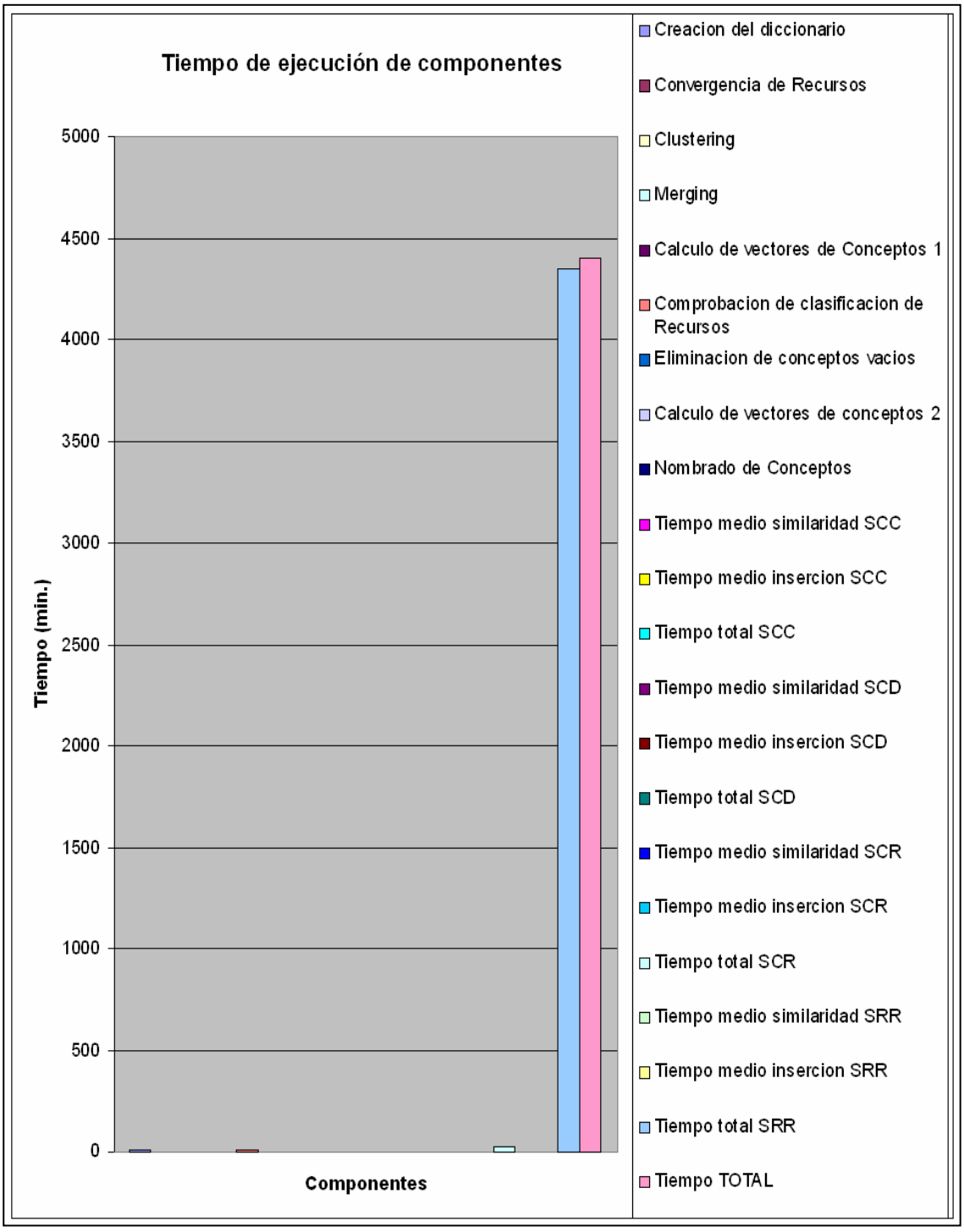


Figura 4.21. Tiempo de Ejecución de Componentes de la Prueba 7.

Otra vez, vemos que la amplia mayoría de tiempo de ejecución ha sido dedicado al cálculo de de las similaridades recurso-recurso.

En la figura 4.22 se pueden ver los concepto y el número de recursos que están clasificados en cada uno. Esta vez vemos que el concepto *ultrasound* no está tan poblado como en otras pruebas debido a que se ha divido en otros conceptos más pequeños como: *baby & ultrasound*, *ultrasound & xray & baby*, etc. También destacan los conceptos de *mri*, *xray*, *radiology* y *pathology*.

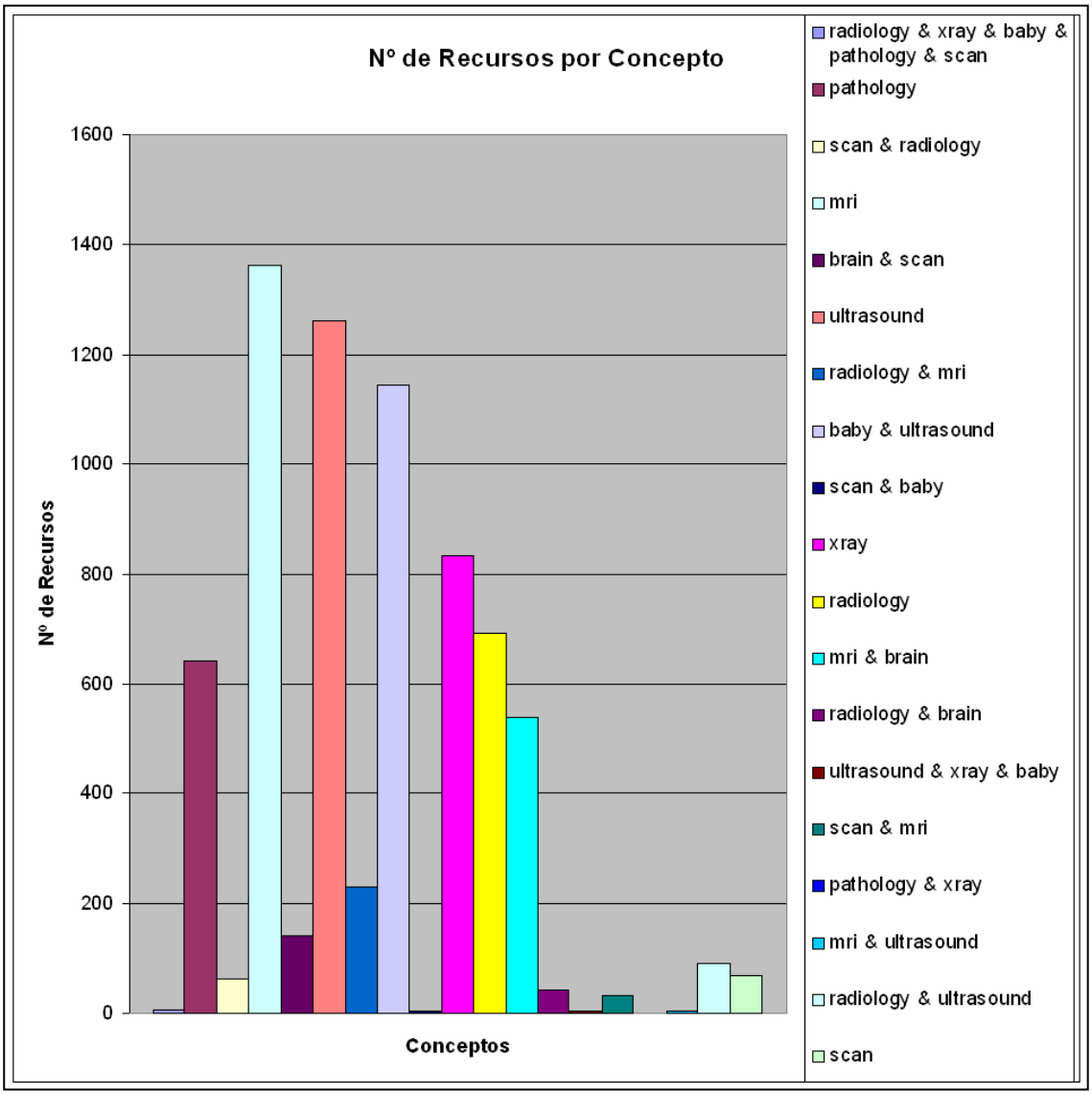


Figura 4.22. Número de Recursos por Concepto de la Prueba 7.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 19.
- Nº de Recursos clasificados: 7148.
- Nº de Recursos no Clasificados: 1431.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 609.

Puede verse en esta prueba que el número de conceptos aparecidos es mayor que en otras cuyo parámetro *dictionaryMina* también está configurado a 500. Esto se debe a que ha subido el valor asignado a *mergingThreshold* a 0.75. Entonces se unen menos conceptos debido a que la condición que lo permite se ha vuelto más restrictiva.

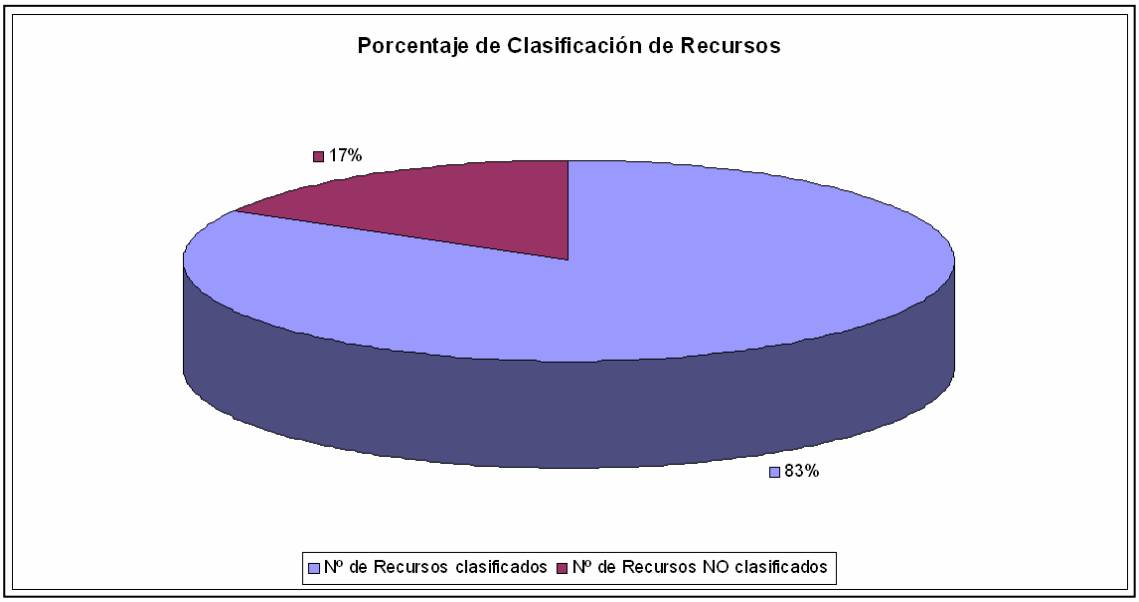


Figura 4.23. Porcentaje de Clasificación de Recursos de la Prueba 7.

La figura 4.23 muestra que en esta prueba también se ha conseguido un porcentaje bastante alto en la clasificación de los recursos, el 83% del total.

4.2.2.1.8. **Prueba 8.**

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.9
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

Del tiempo de ejecución de los componentes hay que destacar lo dicho en el resto de pruebas anteriores, la gran cantidad de tiempo empleado en el cálculo de las similaridades recurso-recurso en relación con el tiempo dedicado a otras funciones del método.

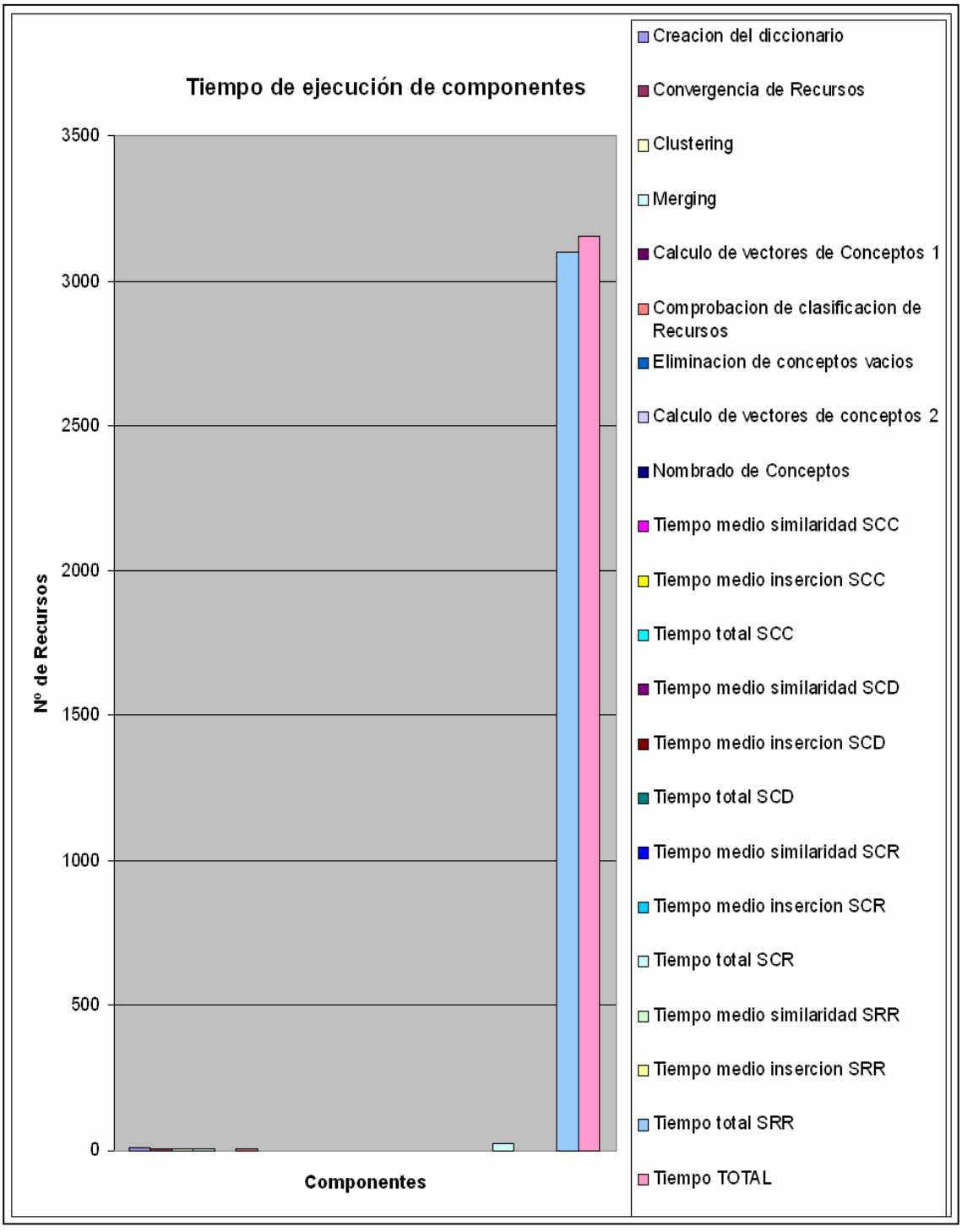


Figura 4.24. Tiempos de Ejecución de Componentes de la Prueba 8.

En la figura 4.25 puede verse que destacan prácticamente los mismos conceptos que en la prueba anterior, y con una variación de población relativamente pequeña.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 18.
- Nº de Recursos clasificados: 7259.
- Nº de Recursos no Clasificados: 1320.

- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 498.

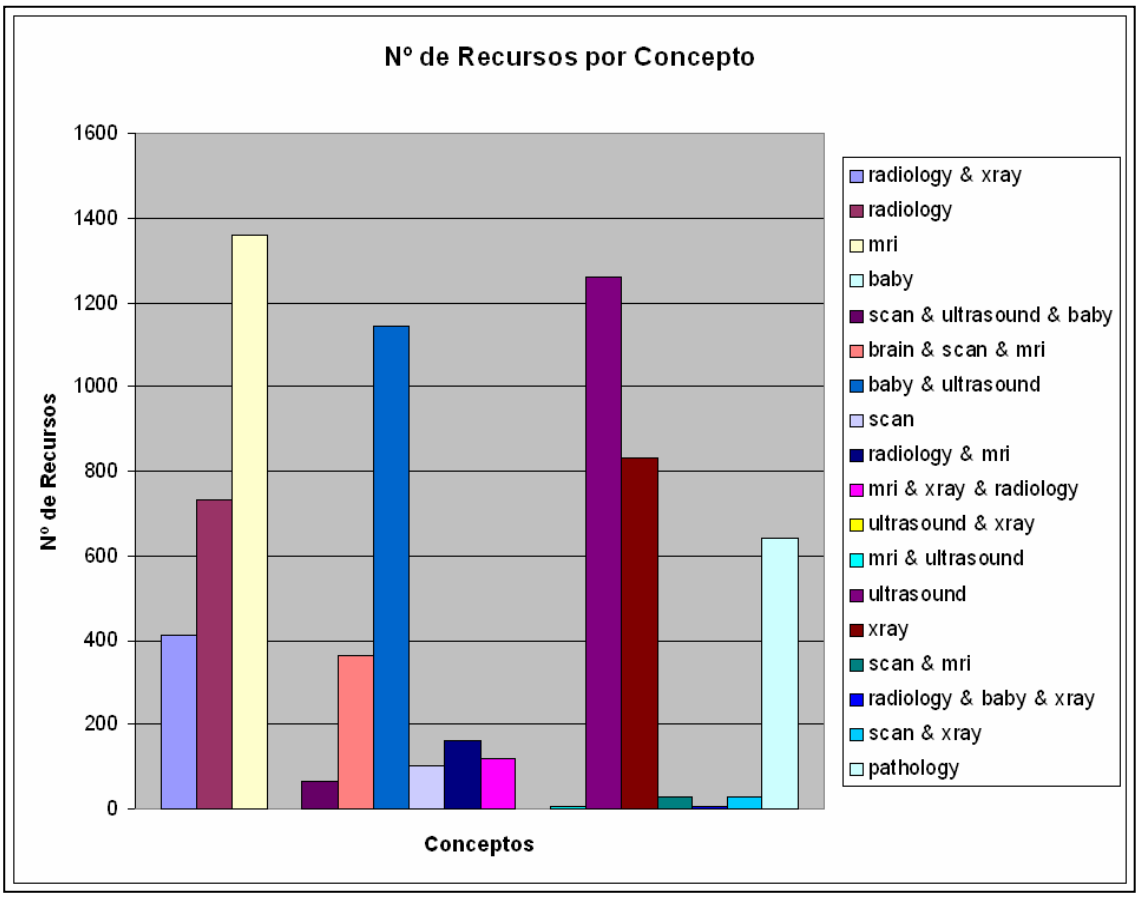


Figura 4.25. Número de Recursos por Concepto de la Prueba 8.

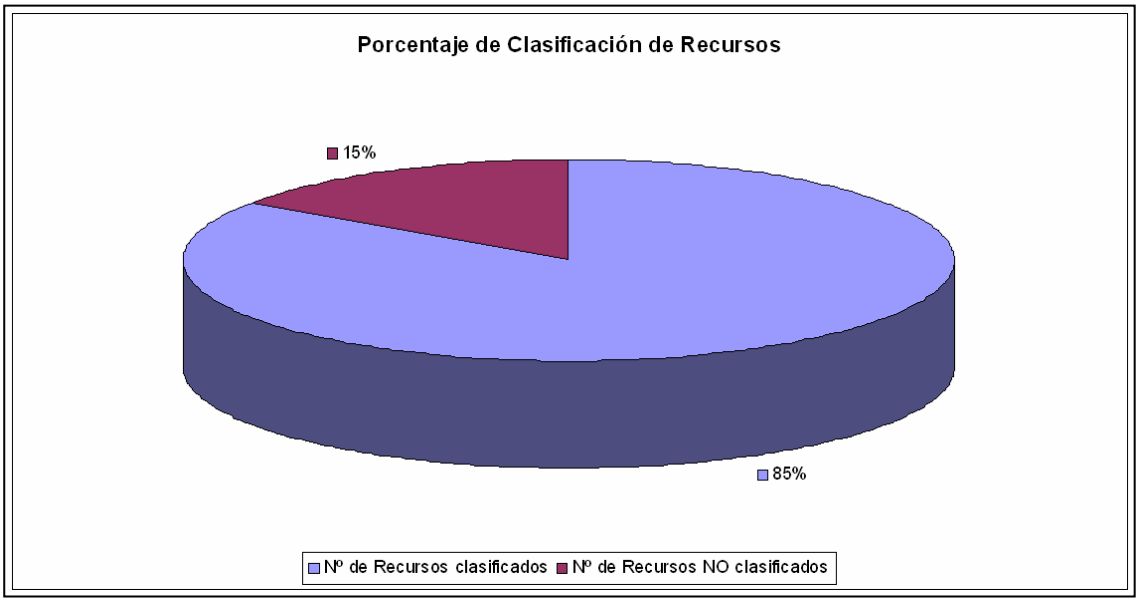


Figura 4.26. Porcentaje de Clasificación de Recursos de la Prueba 8.

En esta prueba se han producido un número menor de conceptos que en la prueba anterior. Esto puede parecer extraño ya que la condición de unión de conceptos se ha definido más estricta en esta prueba. Pero la causa puede ser que se haya generado ya un número menor de conceptos durante el *clustering*, que los creados en la prueba anterior en dicha fase de la ejecución.

En la figura 4.26 puede verse que se ha producido una pequeña mejora en el porcentaje de clasificación de recursos con respecto a la prueba anterior, aumentando hasta el 85%.

4.2.2.1.9. Prueba 9.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.3
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

La figura 4.27 nos muestra los tiempos de ejecución de los componentes para esta prueba, con la clásica diferencia entre el tiempo de ejecución para el cálculo de similaridades recurso-recurso y el resto de componente. Cabe destacar el bajo tiempo empleado para la realización completa de la ejecución, rondando los 295 minutos. Esto es debido a que ha habido pocos cálculos de similaridad recurso-recurso, probablemente por haya habido un porcentaje muy bajo de clasificación de recursos.

Puede observarse en la figura 4.28 el poco número de recursos generados y la poca población de recursos clasificados que tiene cada uno de ellos. Parece que muchos recursos se han quedado sin clasificar.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 6.
- Nº de Recursos clasificados: 1894.
- Nº de Recursos no Clasificados: 6685.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 5863.

Los datos anteriores y los mostrados en la gráfica de la figura 4.28 nos indican lo que ya se sospechaba con anterioridad, que ha habido un porcentaje muy bajo de clasificación de recursos. Esto puede ser debido al aumento del valor de *classifierThreshold* de esta prueba, haciendo que considere muchos recursos clasificados como mal clasificados y los borre del concepto.

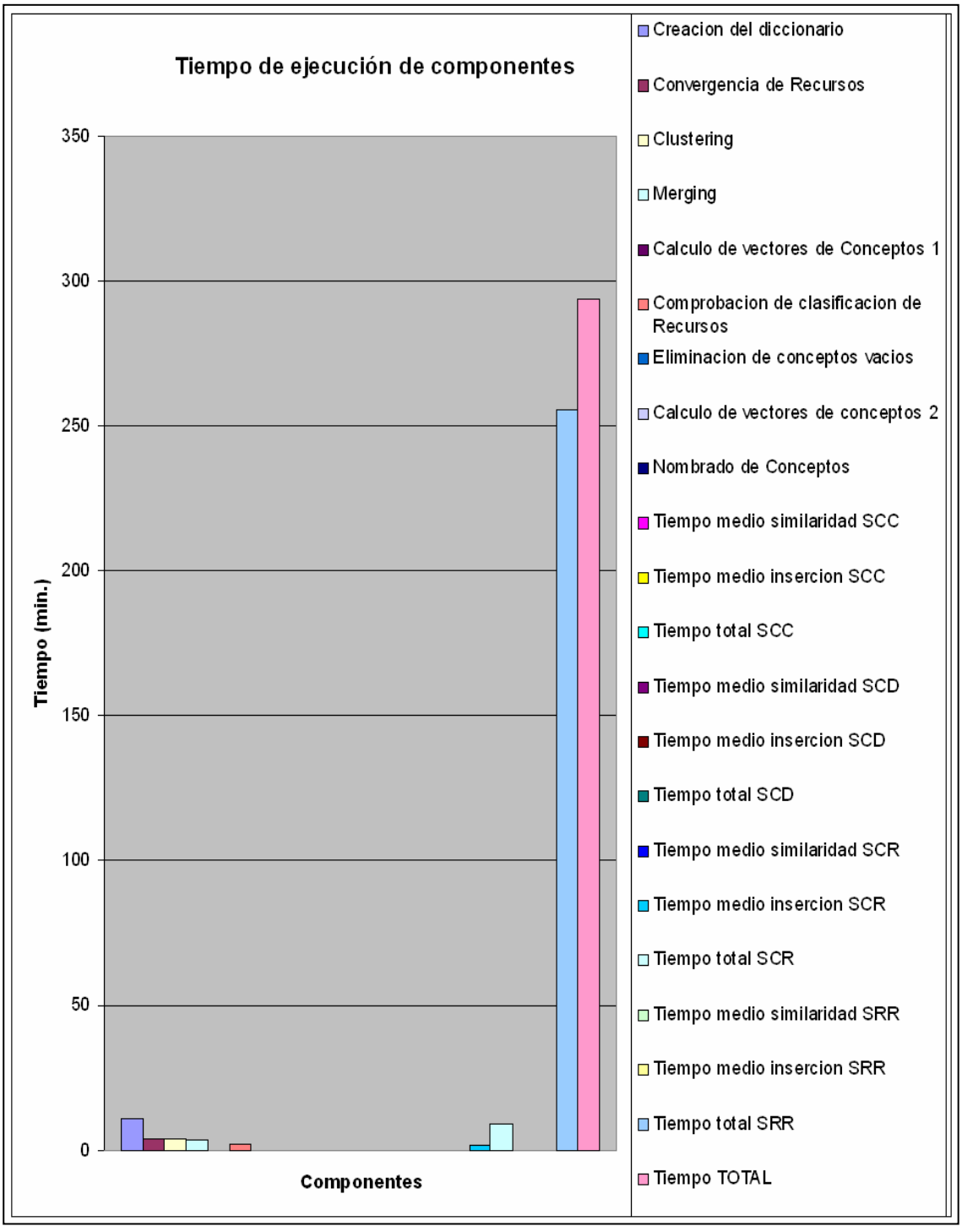


Figura 4.27. Tiempo de Ejecución de Componentes de la Prueba 9.

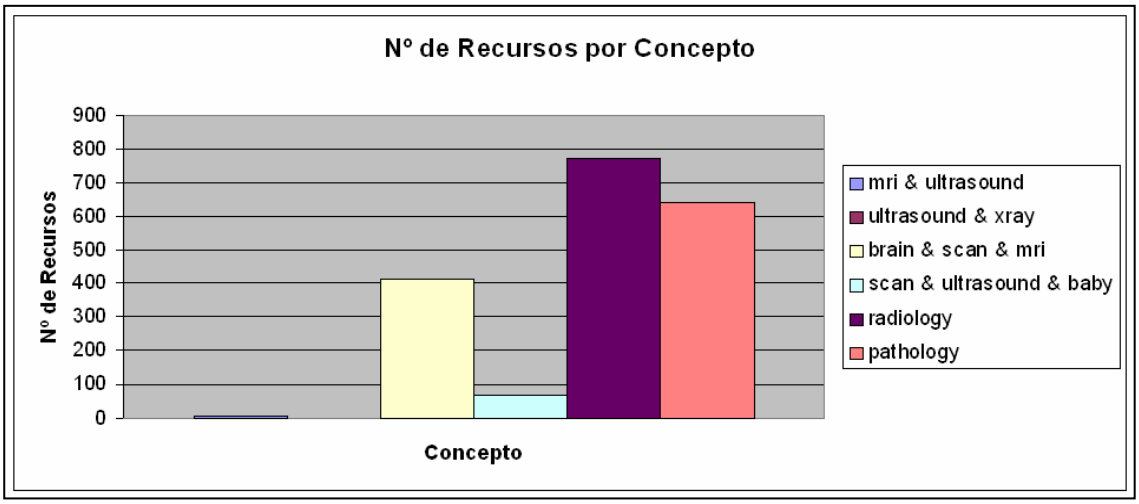


Figura 4.28. Número de Recursos por Concepto de la Prueba 9.

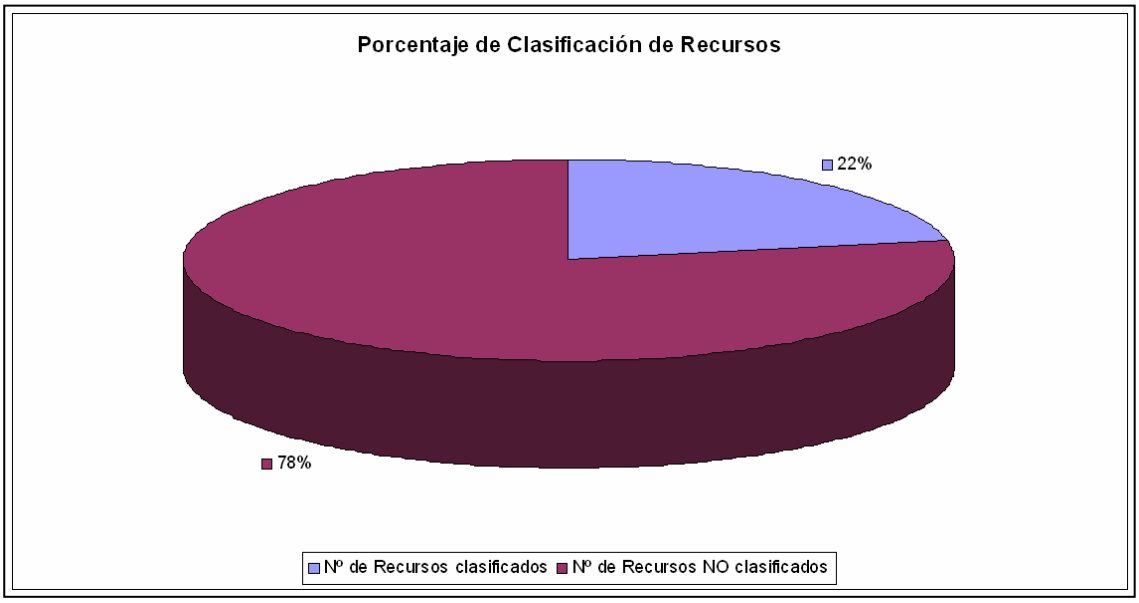


Figura 4.29. Porcentaje de Clasificación de Recursos de la Prueba 9.

4.2.2.1.10. Prueba 10.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.5
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

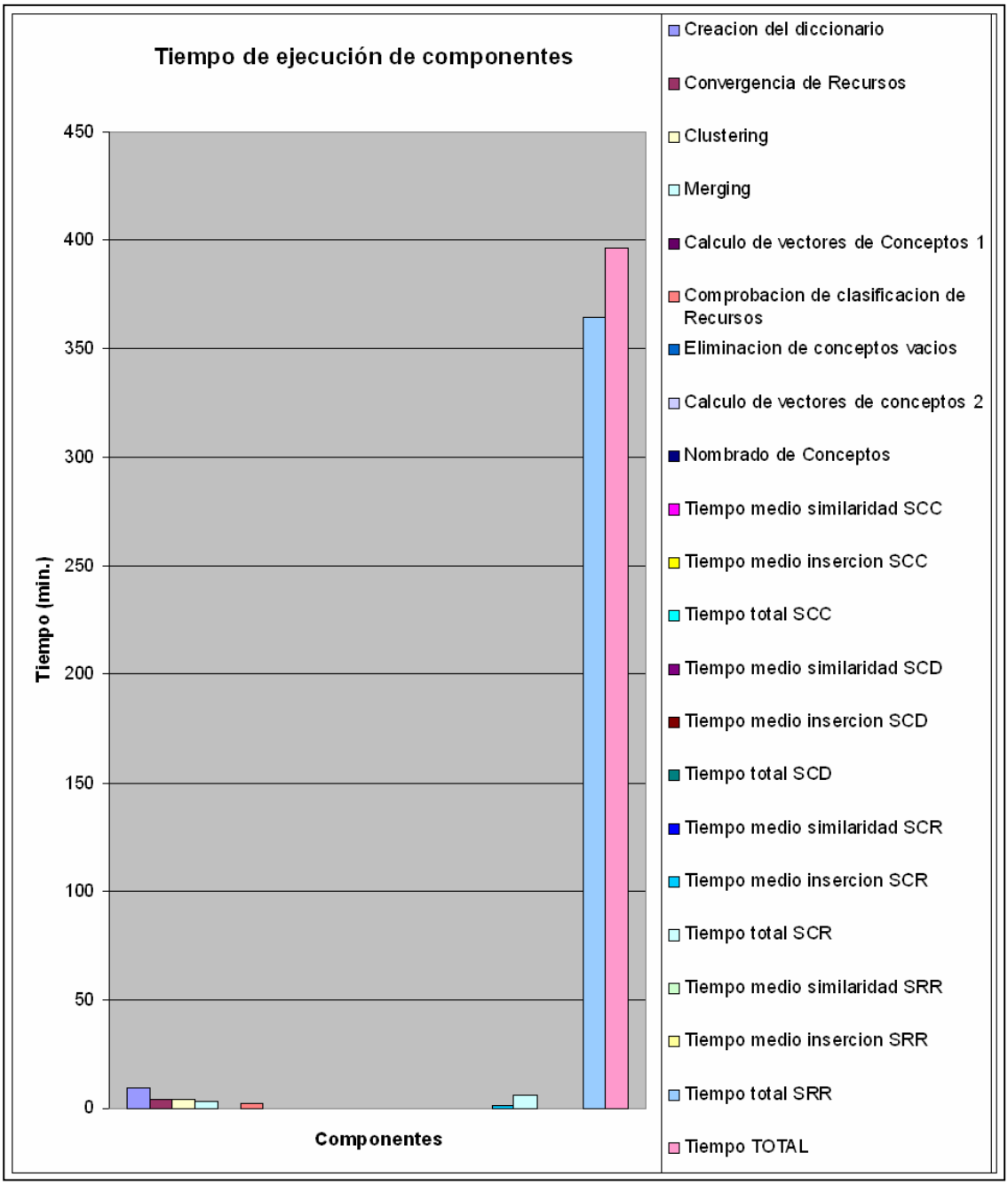


Figura 4.30. Tiempos de Ejecución de Componentes de la Prueba 10.

Otra vez destaca el tiempo de ejecución empleado para el cálculo de similaridades recurso-recurso. También hay que destacar el poco tiempo en el que se ha ejecutado la prueba, rondando los 395 minutos, aunque ha sido superior al de la prueba anterior.

La figura 4.31 indica que se han clasificado pocos recursos en los pocos conceptos que se han generado durante la ejecución de esta prueba de *ACoAR*.

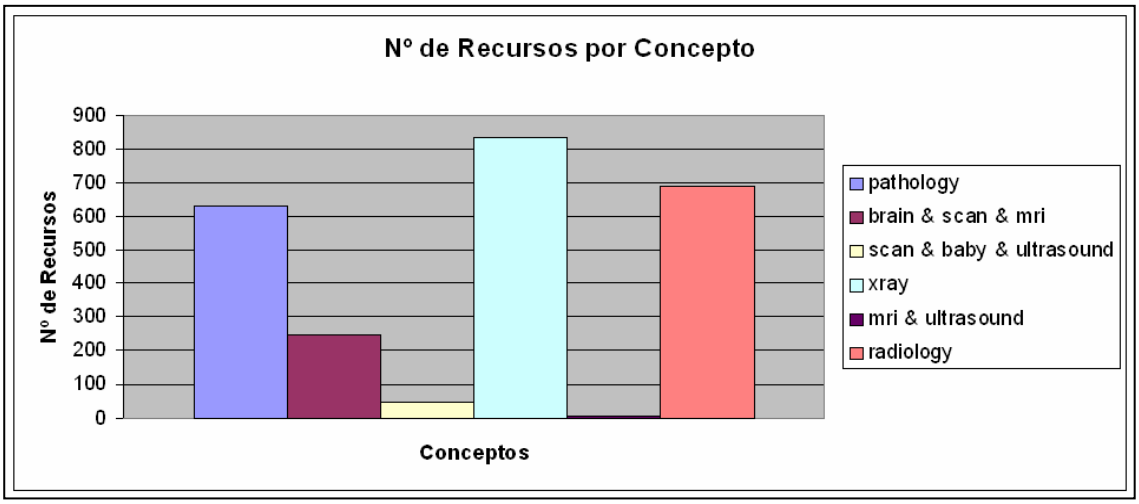


Figura 4.31. Número de Recursos por Concepto de la Prueba 10.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 6.
- Nº de Recursos clasificados: 2455.
- Nº de Recursos no Clasificados: 6124.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 5302.

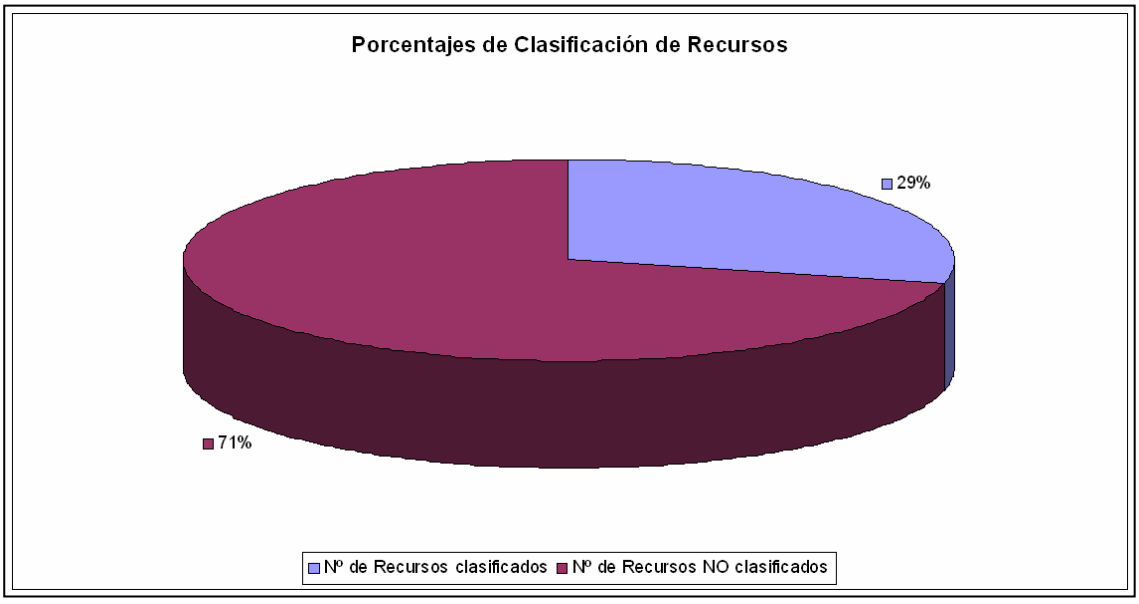


Figura 4.32. Porcentajes de Clasificación de Recursos.

En los datos anteriores se observa que aumentado ligeramente el porcentaje de clasificación de recursos con respecto a la prueba anterior, aunque este sigue siendo muy bajo. En principio, al aumentar, en esta prueba, el valor de *classifierThreshold* era esperado que bajase dicho porcentaje. Pero esto puede haberse producido porque haya habido más recursos clasificados tras realizar el *clustering* que en la prueba anterior.

4.2.2.1.11. Prueba 11.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.2
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

En la gráfica que contiene los tiempos de ejecución de cada componente presentada en la figura 4.33 vuelve a destacar el tiempo empleado para el cálculo de las similaridades recurso-recurso. También puede verse que ha habido una reducción de tiempo en dicho cálculo con respecto a los datos de la prueba 2. Esto puede ser debido al aumento del valor asignado a *similaritiesMinv* o a que haya habido un porcentaje menor de recursos clasificados.

En la figura 4.34 podemos ver que en esta prueba se han generado unos pocos conceptos. Tres de ellos tiene una población relativamente alta y los otros tres, baja.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 6.
- Nº de Recursos clasificados: 6249.
- Nº de Recursos no Clasificados: 2330.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 1508.

Como el porcentaje de recursos clasificados en esta prueba, figura 4.35, es menor que en la prueba 1, no podemos asegurar que la reducción de tiempo en *SRR* se vea afectada por el aumento del valor asignado a *similaritiesMinv*.

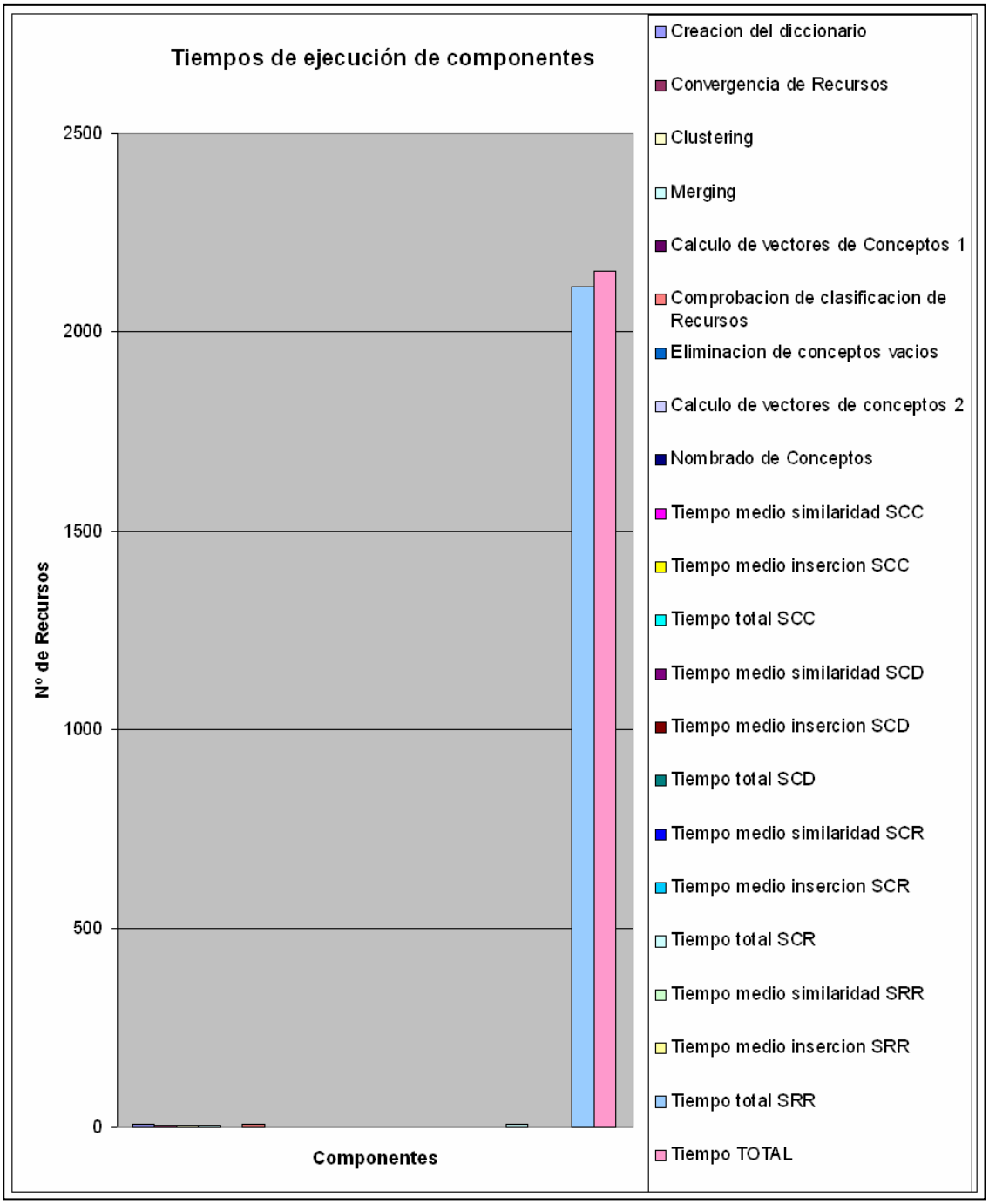


Figura 4.33. Tiempos de Ejecución de Componentes de la Prueba 11.

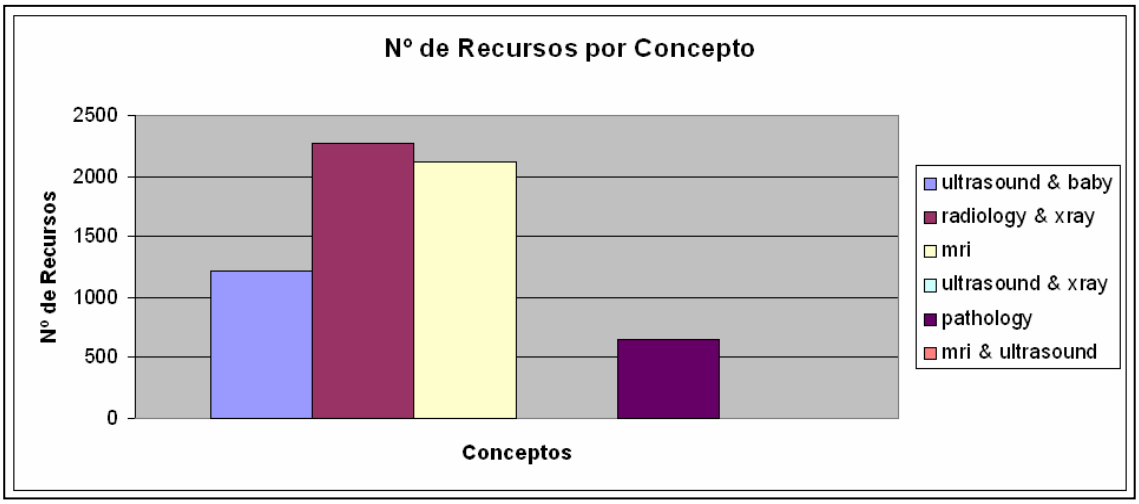


Figura 4.34. Número de Recursos por Concepto de la Prueba 11

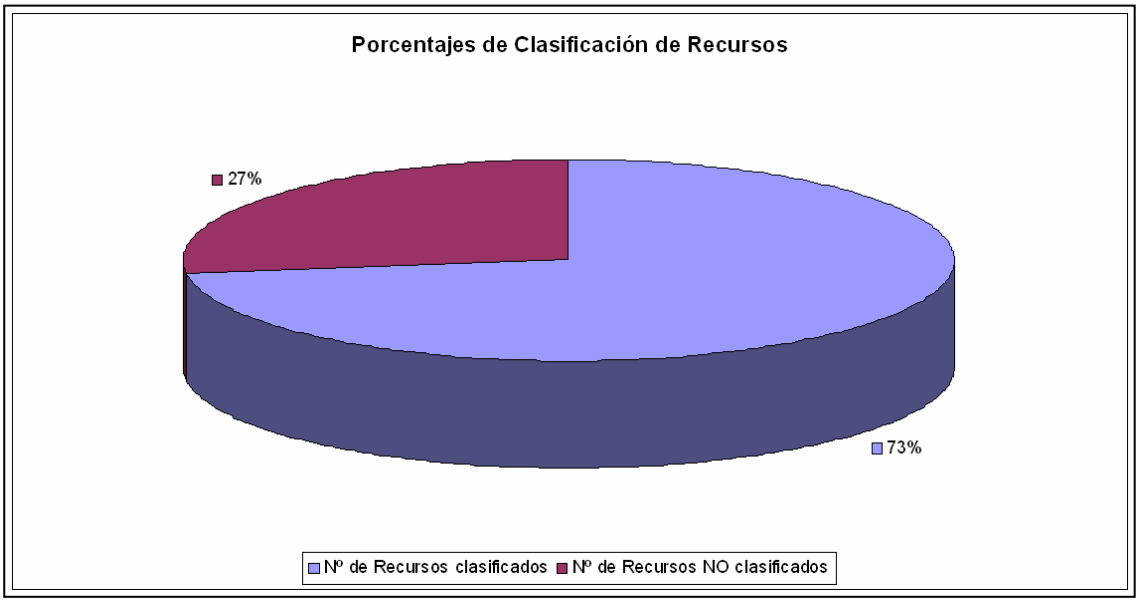


Figura 4.35. Porcentajes de Clasificación de Recursos de la Prueba 11.

4.2.2.1.12. Prueba 12.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.5
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

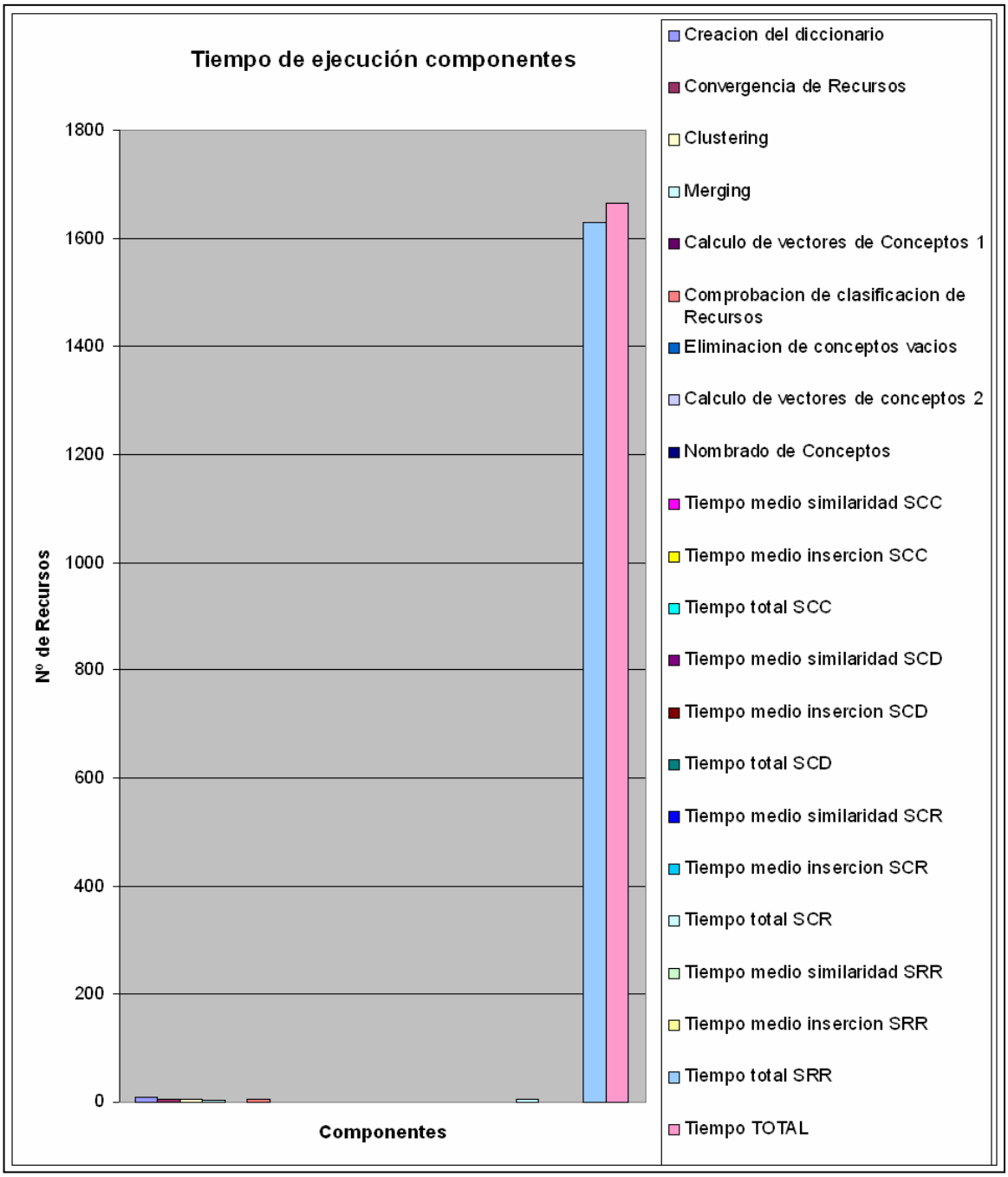


Figura 4.36. Tiempo de Ejecución de Componentes de la Prueba 12.

Otra vez la mayor parte de l tiempo de ejecución ha estado ocupado con el cálculo de las similaridades recurso-recurso. En esta prueba también ha habido una reducción de tiempo de ejecución en comparación con la prueba 2, pero puede que también sea porque halla menos recursos clasificados en vez de que sea causa del aumento del valor asignado al parámetro *similaritiesMinv*.

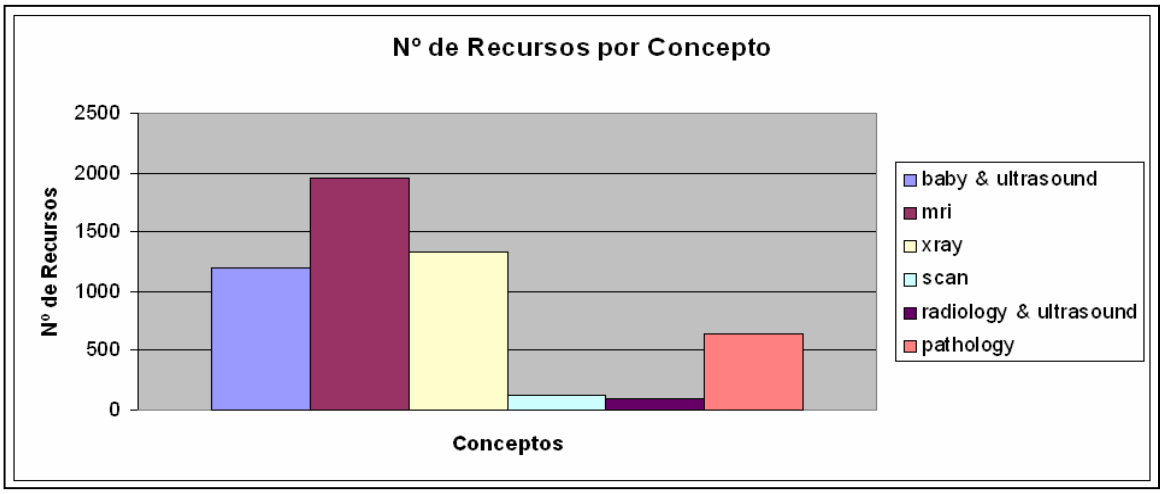


Figura 4.37. Número de Recursos por Concepto de la Prueba 12.

En la figura 4.37 puede observarse el número de recursos por concepto de esta prueba. Esta vez el concepto más poblado es *mri*, en vez de *ultrasound* como en gran parte de otras pruebas. Probablemente aquellos recursos anteriormente clasificados como ecografías hallan sido clasificados en otros conceptos, o pueden que hallan sido eliminados del concepto al que pertenecían por ser considerados como mal clasificados.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 6.
- Nº de Recursos clasificados: 5328.
- Nº de Recursos no Clasificados: 3251.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 2429.

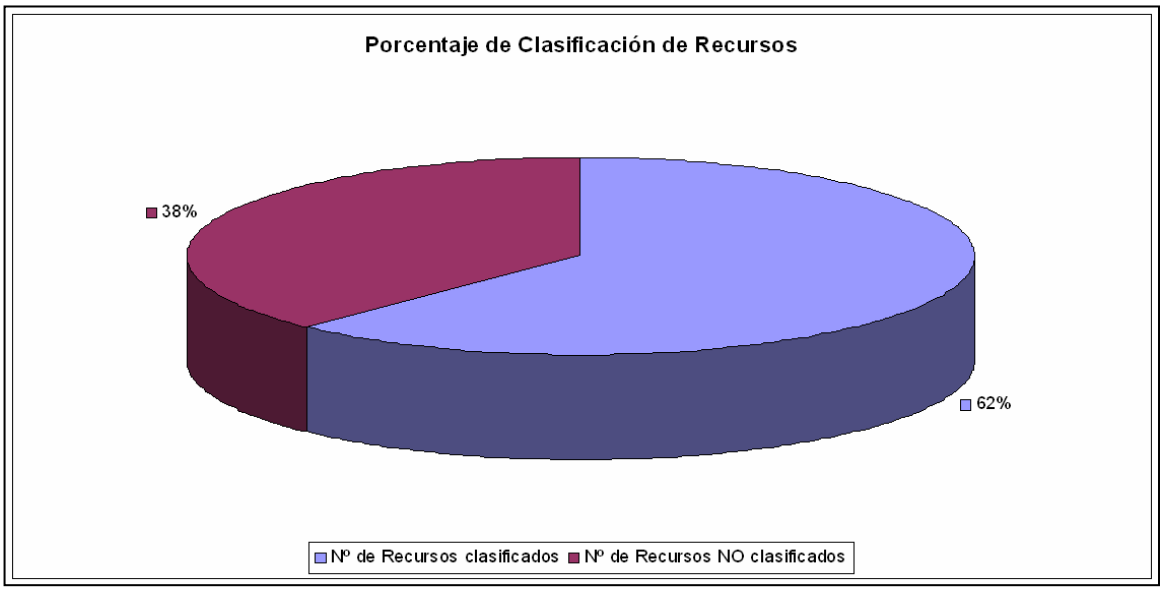


Figura 4.38. Porcentaje de Clasificación de Recursos de la Prueba 12.

En esta prueba también vemos que ha habido un porcentaje menor de recursos clasificados. Por tanto la reducción del tiempo de ejecución de la aplicación con respecto a la prueba 2, es muy probable que sea debido a esta reducción del porcentaje de recursos clasificados.

4.2.2.1.13. Prueba 13.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.75
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

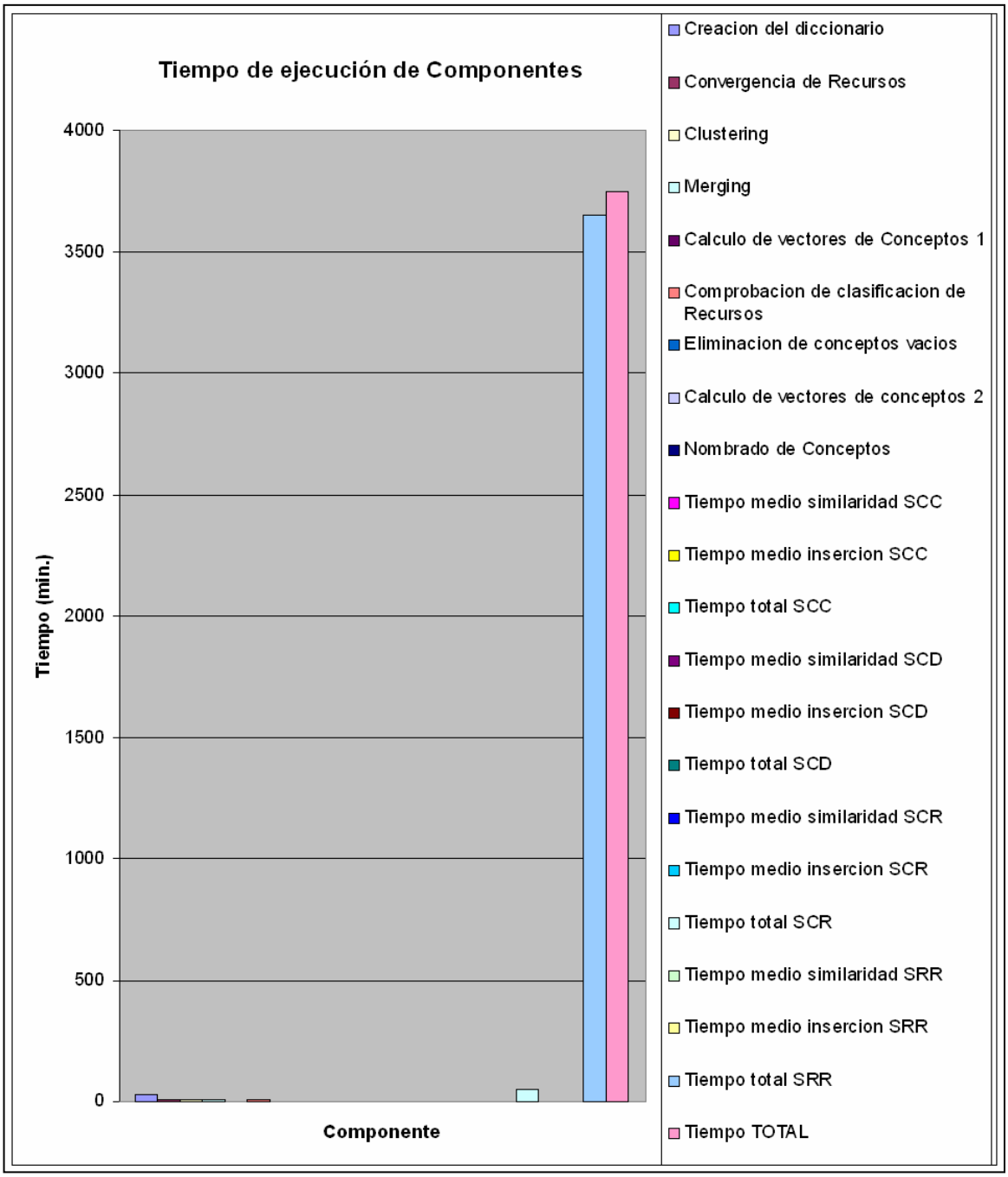


Figura 4.39. Tiempo de Ejecución de Componentes de la Prueba 13.

En esta prueba también destaca el tiempo empleado para el cálculo de SRR con respecto al resto de módulos.

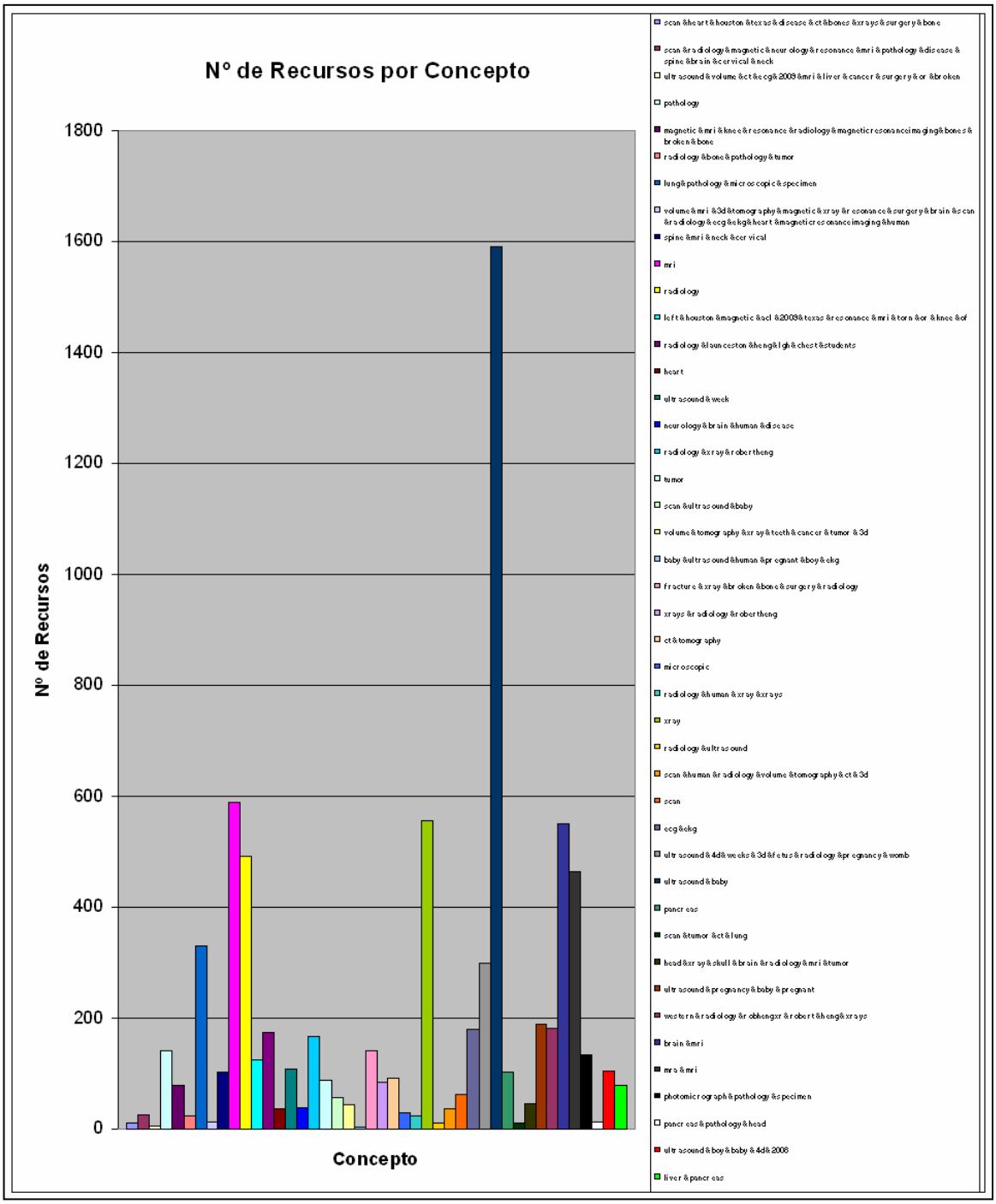


Figura 4.40. Número de Recursos por Concepto de la Prueba 13.

Al haber reducido el valor del parámetro *dictionaryMina*, nos encontramos con un gran número de conceptos, siendo "*ultasound & baby*" el que destaca debido al número de recursos que están asociados a él.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 44.
- Nº de Recursos clasificados: 7613.
- Nº de Recursos no Clasificados: 966.
- Nº de Recursos *Pending*: 173.

- Nº de Recursos *Converged*: 793.

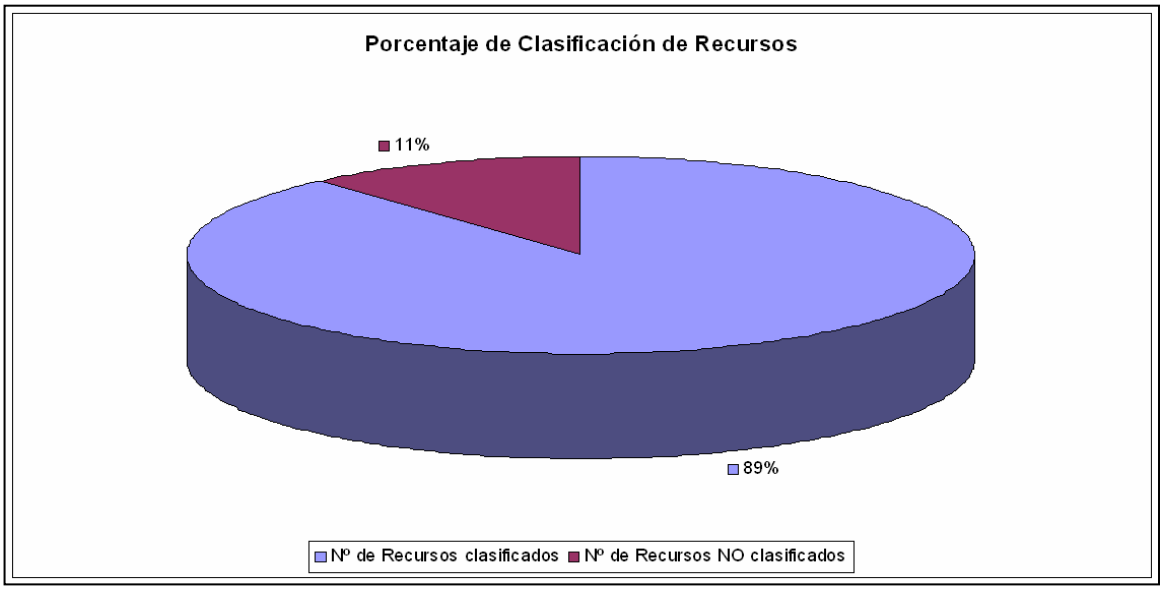


Figura 4.41. Porcentaje de Clasificación de Recursos de la Prueba 13

En la figura 4.41 puede verse que se ha conseguido un porcentaje de clasificación de los recursos bastante alto, un 89%.

4.2.2.1.14. Prueba 14.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 2564.
- *convergenceMina*: 1
- *mergingThreshold*: 0.75
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

En la gráfica de los tiempos de ejecución de componentes, mostrada en la figura 4.42, vuelve a destacar el tiempo de ejecución del módulo del cálculo de *SRR*.

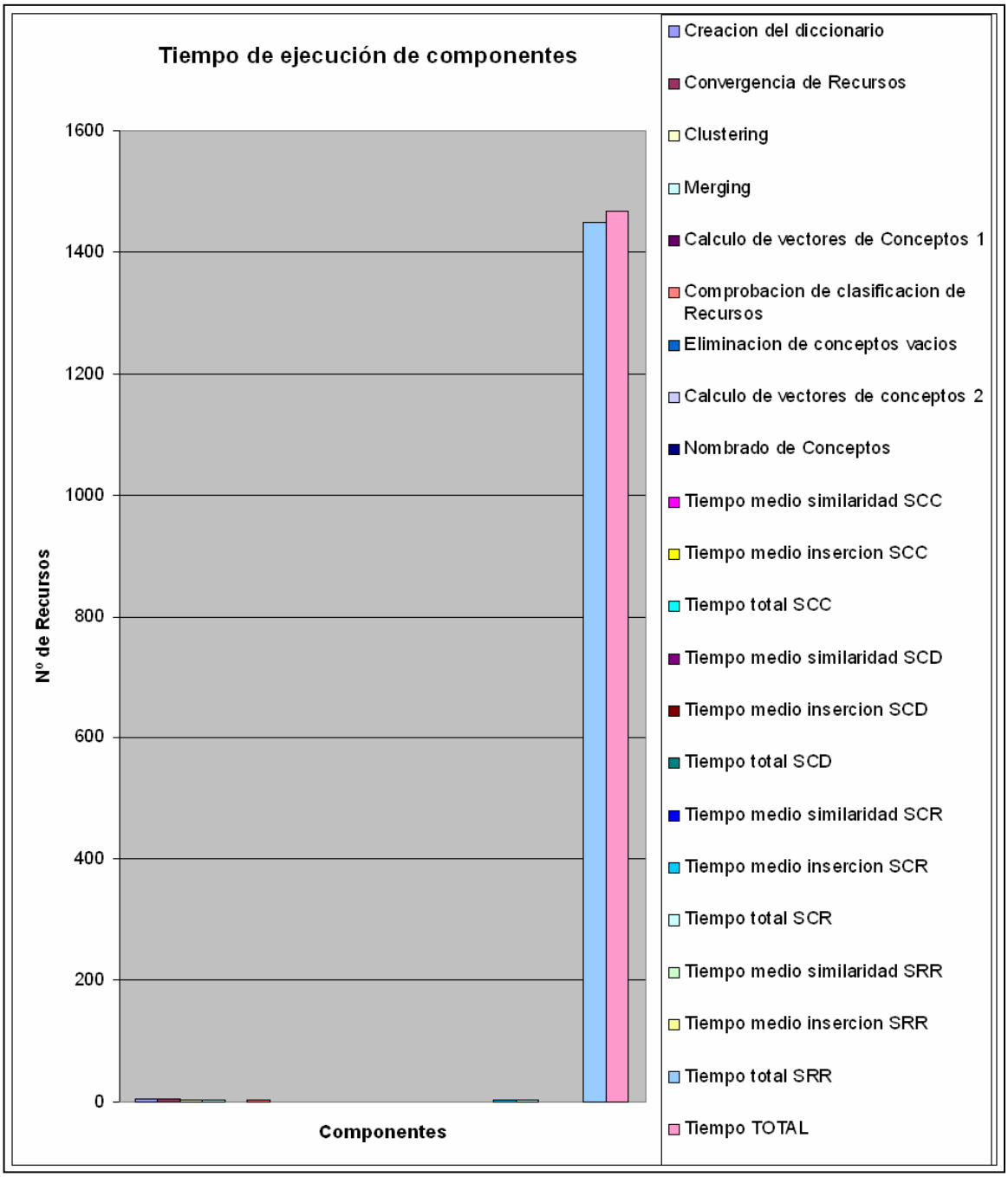


Figura 4.42. Tiempos de Ejecución de Componentes de la Prueba 14.

En esta prueba, al estar el parámetro *dictionaryMina* configurado con el valor 2564, solamente se ha creado un concepto, el de *ultrasound*, con sólo una etiqueta, la que da nombre al concepto, y sólo se han clasificado aquellos recursos con dicha etiqueta.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 1.
- Nº de Recursos clasificados: 2564.
- Nº de Recursos no Clasificados: 6015.
- Nº de Recursos *Pending*: 6015.

- Nº de Recursos *Converged*: 0.

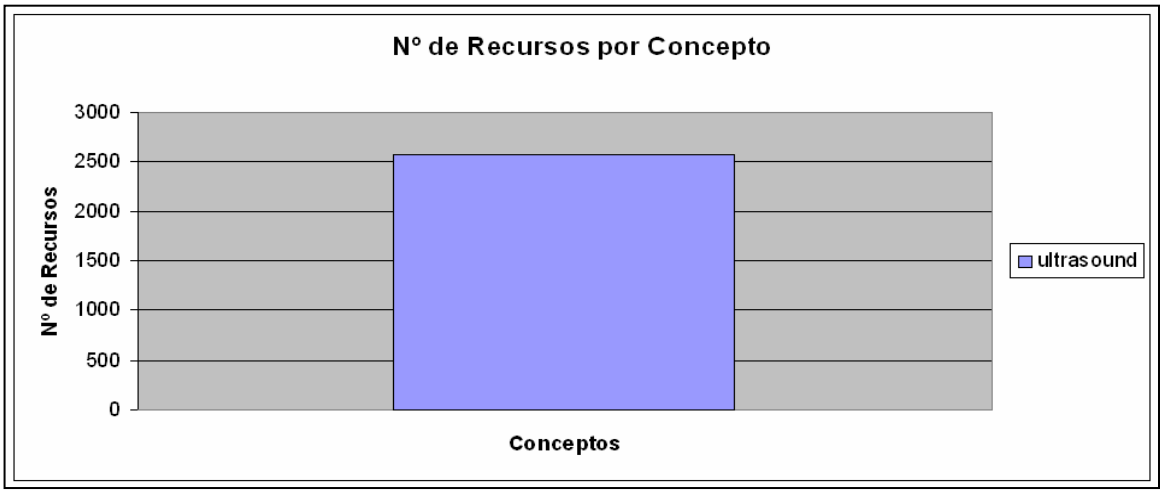


Figura 4.43. Número de Recursos por concepto de la Prueba 14.

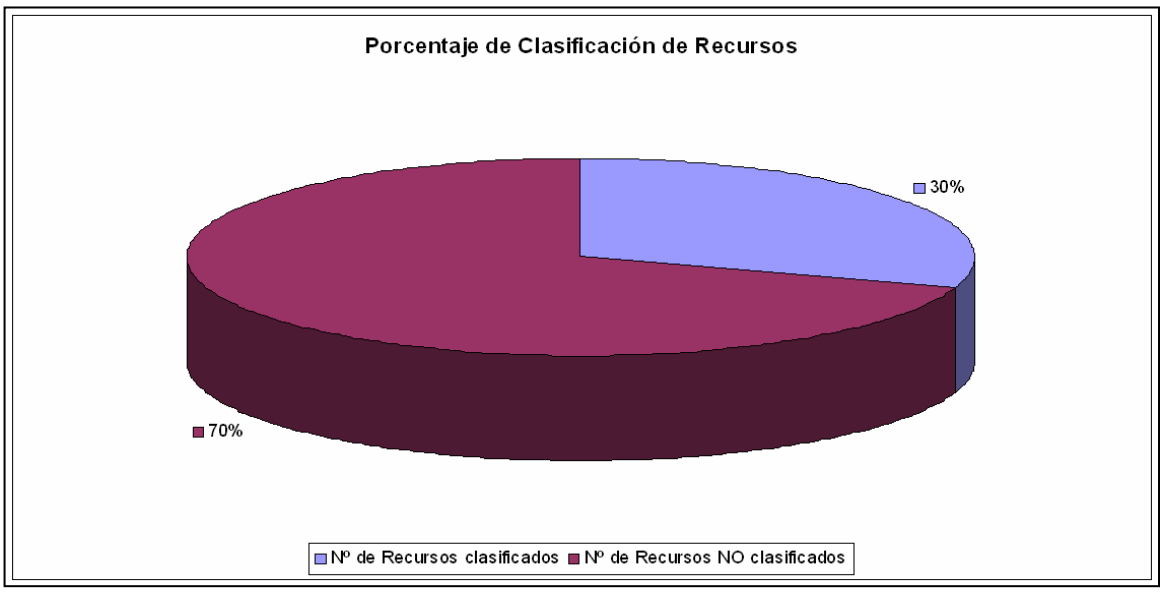


Figura 4.44. Porcentaje de Clasificación de Recursos de la Prueba 14.

Puede observarse en la figura 4.44, como en otras pruebas con el parámetro *dictionayMina* configurado a 2564, que la gran mayoría de recursos ha quedado sin clasificar, el 70%.

4.2.2.1.15. Prueba 15.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.9
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1

- *similaritiesSRRThres*: 0.4
- *classifierT*: delta.

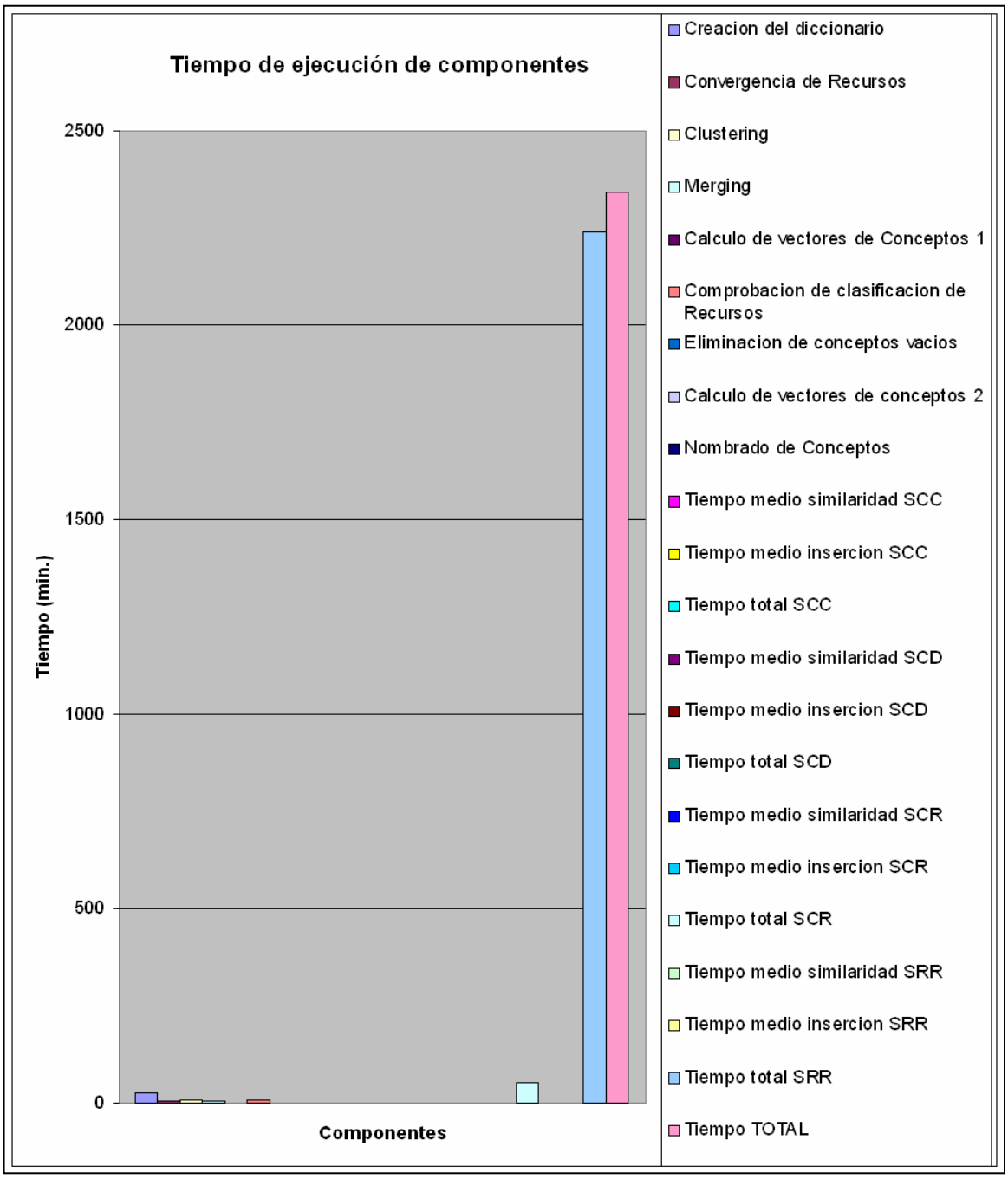


Figura 4.45. Tiempo de Ejecución de Componentes de la Prueba 14.

Puede verse que esta prueba también sigue la norma general de las anteriores, usando la mayor parte del tiempo de ejecución para el cálculo de las simiaridades recurso-recurso.

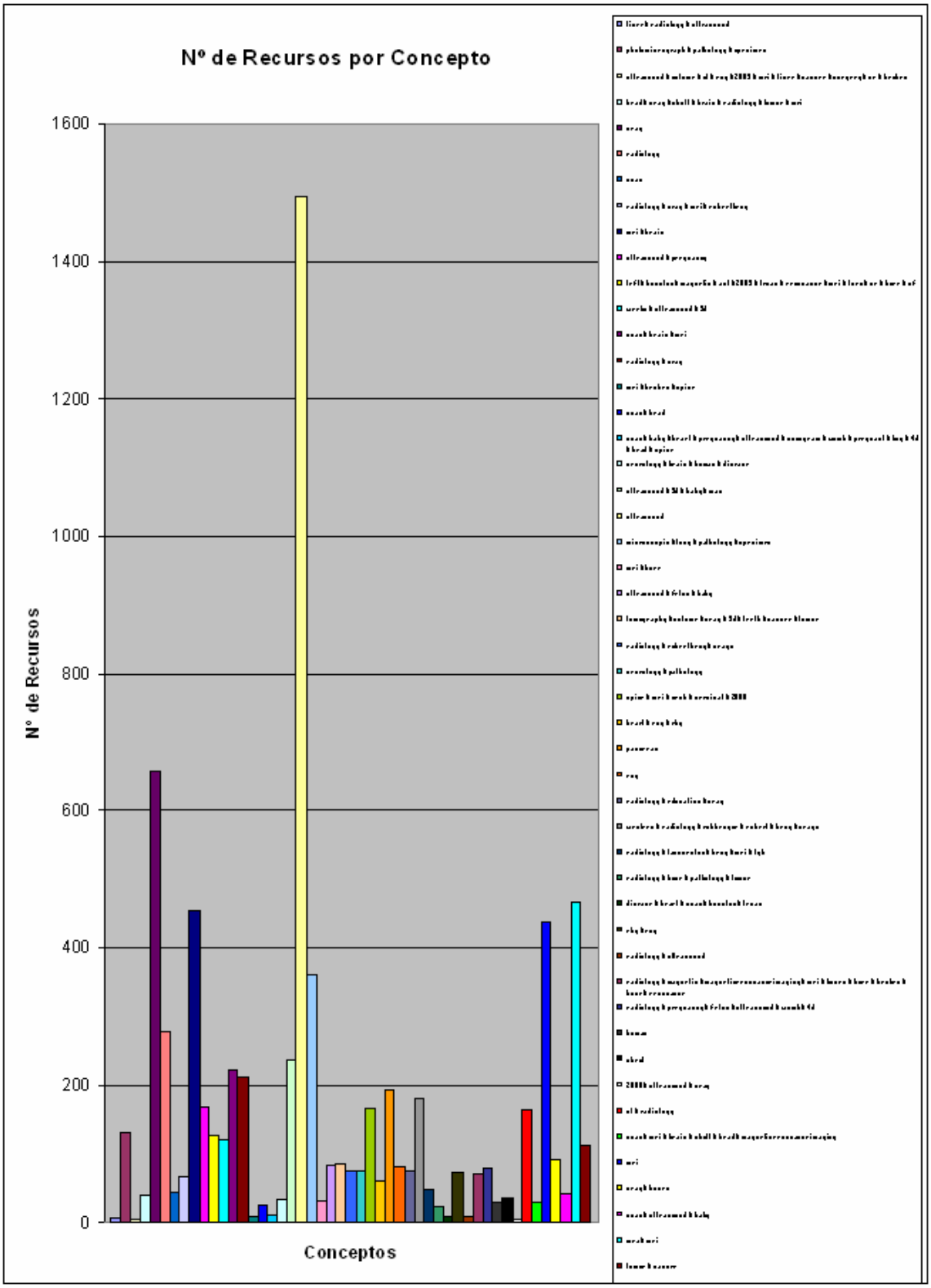


Figura 4.46. Número de Recursos por Concepto de la Prueba 15.

En la figura 4.46 puede apreciarse que el concepto con mayor población es *ultrasound*. Además, se ve que se han generado una gran cantidad de conceptos. A continuación se detalla dicha cantidad.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 49.
- Nº de Recursos clasificados: 7529.
- Nº de Recursos no Clasificados: 1050.
- Nº de Recursos *Pending*: 173.
- Nº de Recursos *Converged*: 877.

Los datos anteriores muestran que ha aumentado el número de conceptos creados con respecto a los datos de la prueba 13. Esto es, presumiblemente, debido al aumento del valor asignado al parámetro *mergingThreshold*, lo que hace más estricta la condición para unir dos conceptos similares.

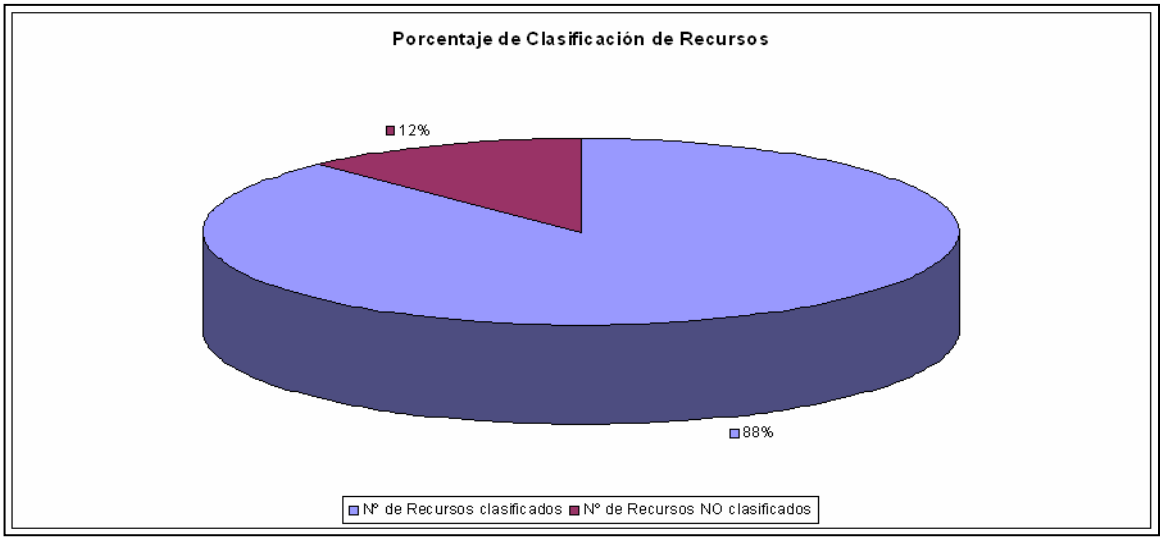


Figura 4.47. Porcentaje de Clasificación de Recursos de la Prueba 15.

En esta prueba se ha conseguido un porcentaje bastante elevado de recursos clasificados, rondando el 88%. Una cifra muy similar a la de la prueba 13, aunque la empeora ligeramente.

4.2.2.1.16. Prueba 16.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.3
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta

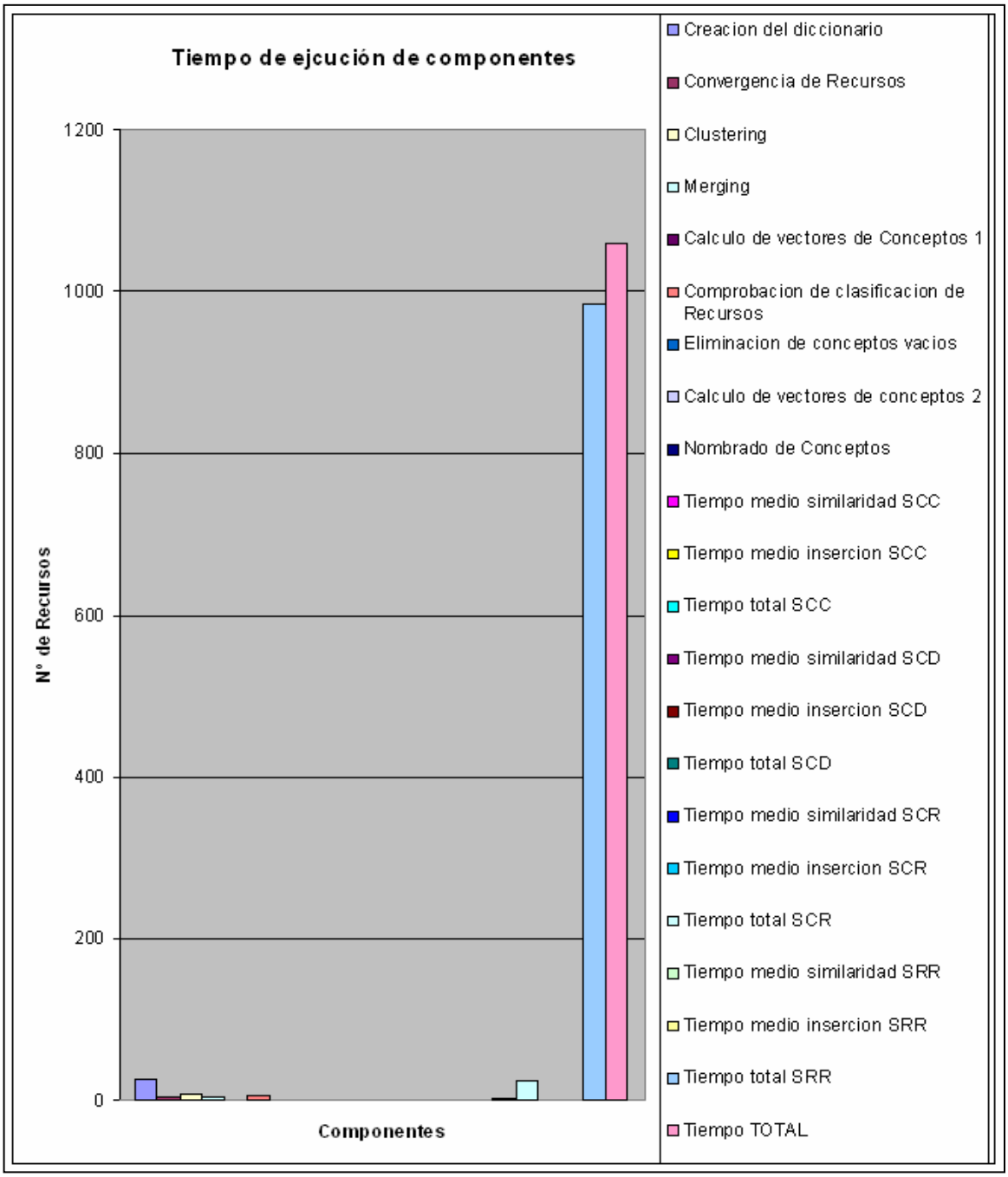


Figura 4.48. Tiempos de Ejecución de los Componentes de la Prueba 16.

Otra vez, el máximo tiempo empleado en la ejecución es para el cálculo de SRR. Hay que fijarse que el tiempo total es más bajo que otras pruebas, como la 15 y la 13. Esto, probablemente, sea debido a que se hayan clasificado menos recursos a causa del aumento del valor del parámetro *classifierThreshold*.

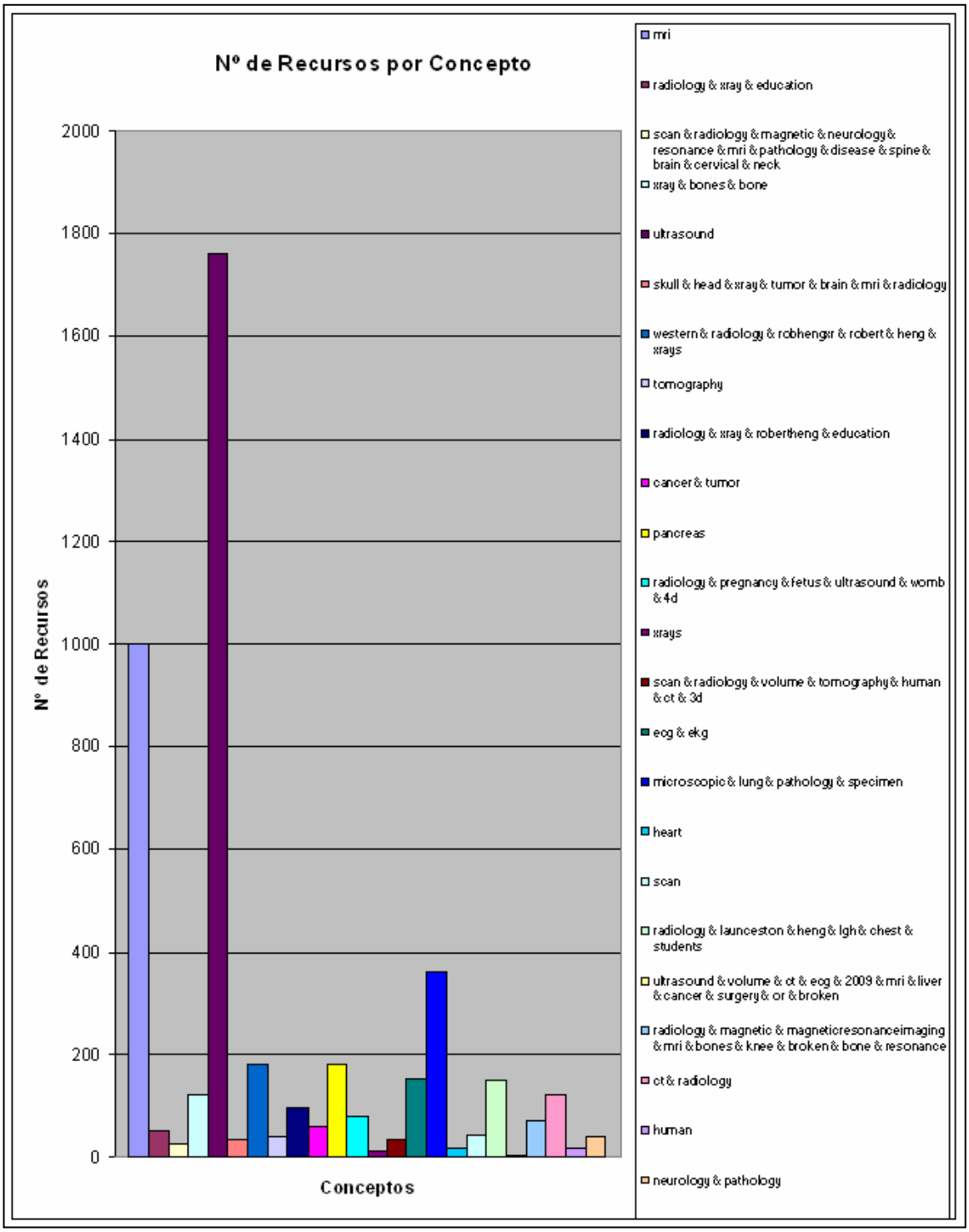


Figura 4.49. Número de Recursos por Concepto de la Prueba 16

En la figura 4.49 puede observarse que los conceptos dominantes de esta prueba son *ultrasound*, *mri*, y de forma más humilde "*microscopic & lung & pathology & specimen*".

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 24.
- Nº de Recursos clasificados: 4664.
- Nº de Recursos no Clasificados: 3915.

- Nº de Recursos *Pending*: 173.
- Nº de Recursos *Converged*: 3742.
-

En esta prueba se han generado menos recursos respecto con las pruebas inmediatamente anteriores, con el parámetro *dictionaryMina* configurado a 100 anotaciones. Esto puede estar directamente relacionado con el, también más bajo, porcentaje de recursos clasificados, y a su vez por el aumento del valor de *classifierThreshold*.

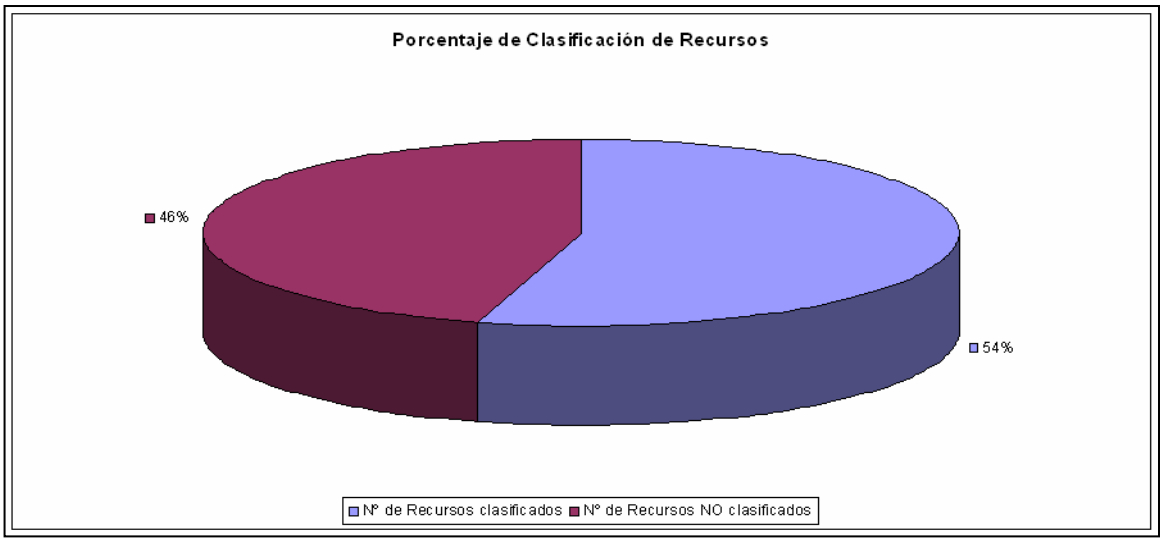


Figura 4.50. Porcentaje de Clasificación de Recursos de la Prueba 16.

4.2.2.1.17. Prueba 17.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.5
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta

Los tiempos de ejecución de esta prueba, mostrados en la figura 4.51, indican algo diferente a otras pruebas. El tiempo empleado para el cálculo de SRR solamente ronda un 60% del tiempo total de ejecución de la aplicación. Además, dicho tiempo también es extremadamente bajo comparado con la gran mayoría del resto de pruebas. Esto, probablemente, sea un indicativo de un porcentaje muy bajo de clasificación, relacionado con el aumento del valor asignado a *classifierThreshold*.

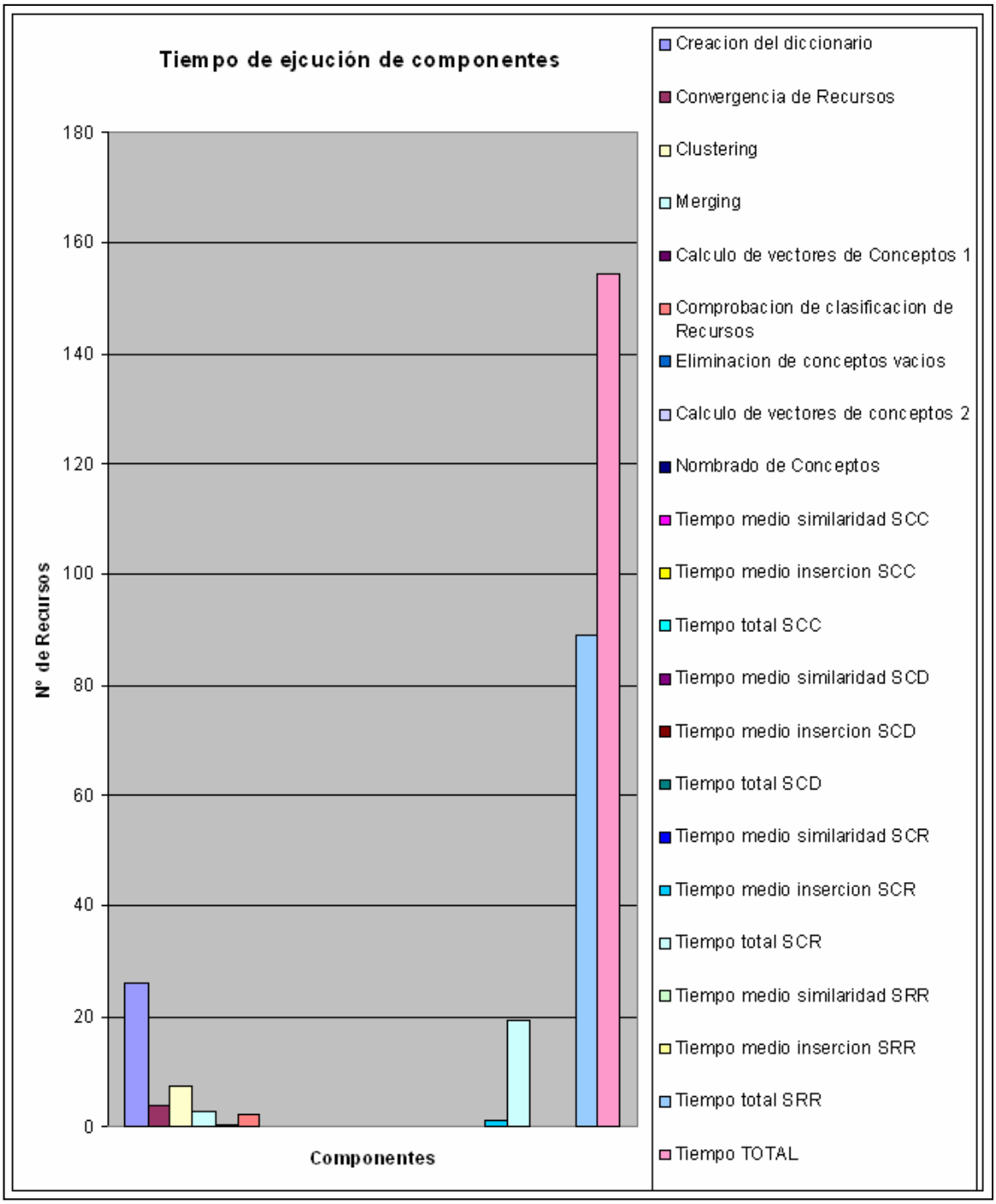


Figura 4.51. Tiempo de Ejecución de Componentes de la Prueba 17.

Los datos mostrados en la figura 4.52 indican que el concepto con más recursos clasificados es "*mra & mri*". Cabe destacar que los conceptos relacionados con *ultrasound*, muy poblados en gran parte de otras pruebas, aquí tienen unas poblaciones reducidas.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 20.
- Nº de Recursos clasificados: 1765.
- Nº de Recursos no Clasificados: 6814.
- Nº de Recursos *Pending*: 173.

- Nº de Recursos *Converged*: 6641.

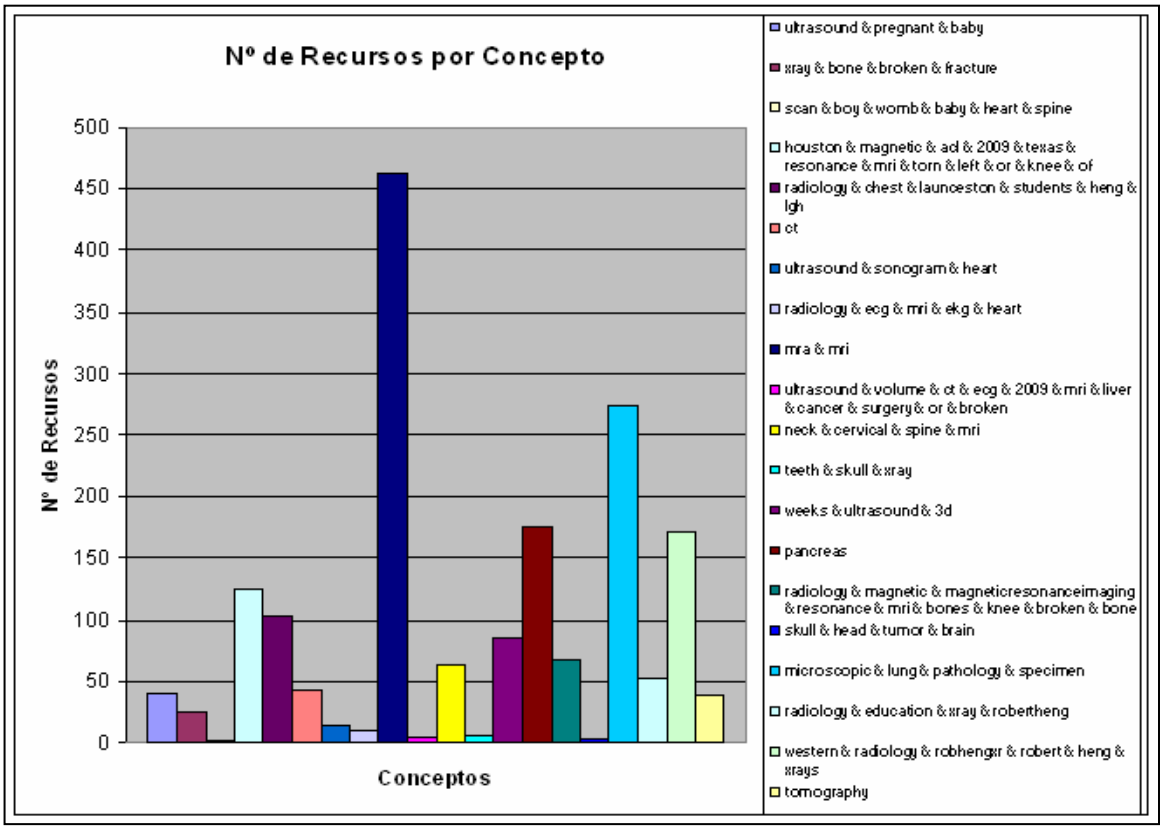


Figura 4.52. Número de Recursos por Concepto de la Prueba 17.

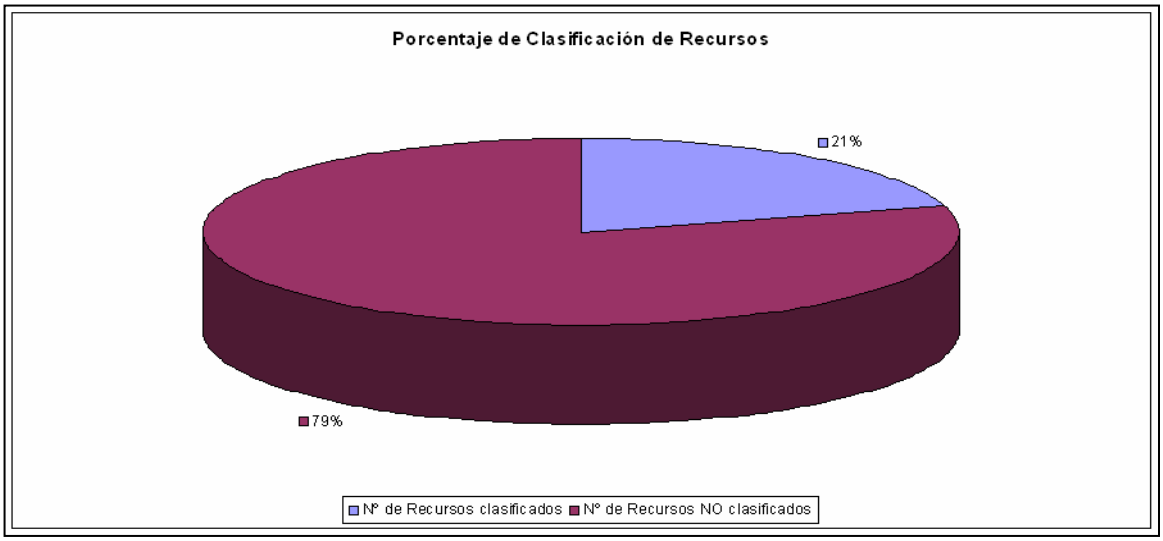


Figura 4.53. Porcentaje de Clasificación de Recursos de la Prueba 17.

Como se esperaba con los datos anteriores, en la figura 4.53, aparece un porcentaje de clasificación bajo. Debido a esto también se ha reducido el número de conceptos generados con respecto a la prueba 16.

4.2.2.1.18. Prueba 18.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.2
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta

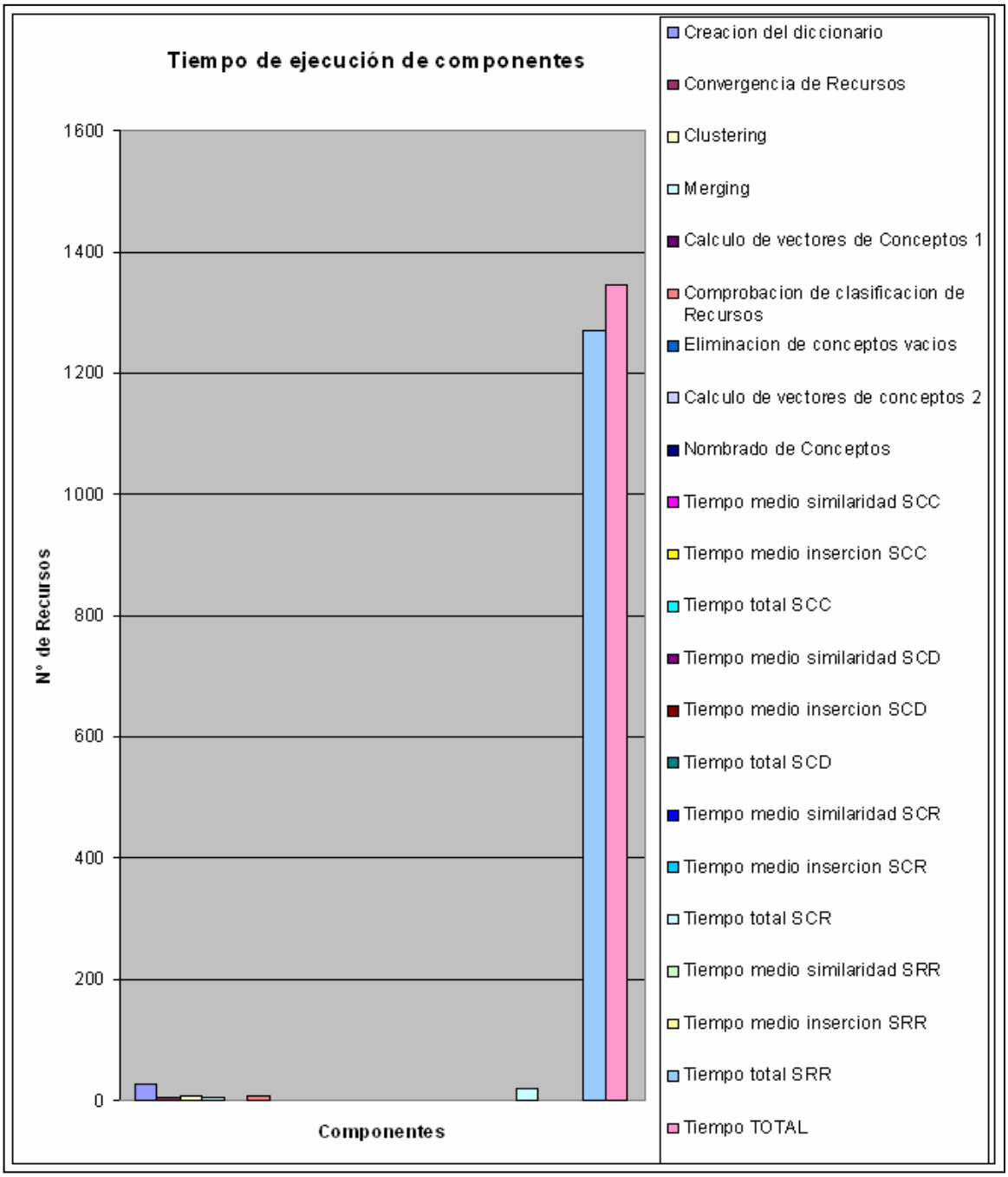


Figura 4.54. Tiempo de Ejecución de los Componentes de la Prueba 18.

Esta prueba vuelve a tomar la tendencia de usar la mayor parte del tiempo de ejecución de la misma para el cálculo de las similaridades recurso-recurso.

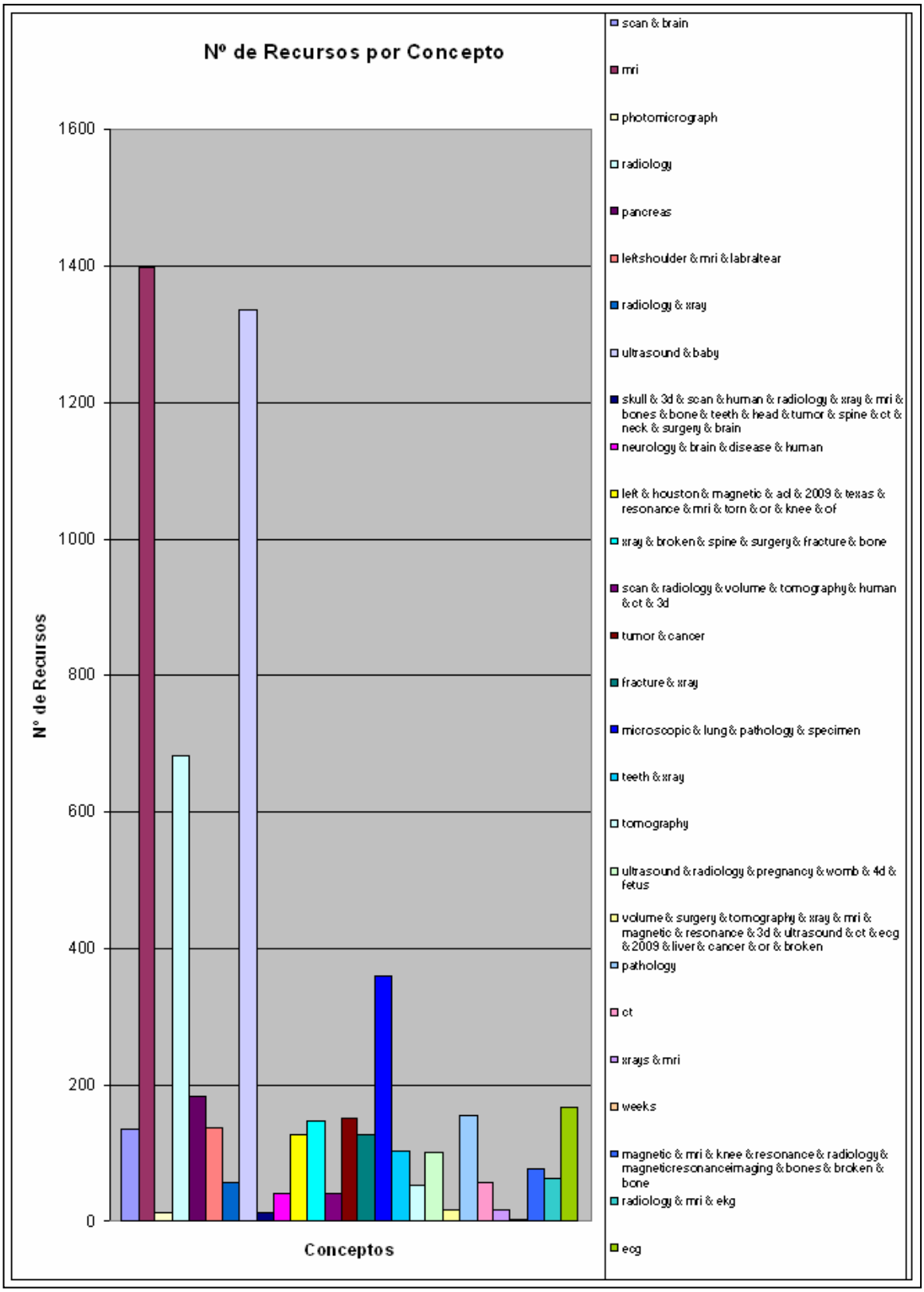


Figura 4.55. Número de Recursos por Concepto de la Prueba 18.

En esta prueba los conceptos con más recursos clasificados en ellos son: *mri*, "*ultrasound & baby*" y *radiology*.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 27.
- Nº de Recursos clasificados: 5754.
- Nº de Recursos no Clasificados: 2825.
- Nº de Recursos *Pending*: 173.
- Nº de Recursos *Converged*: 2652.

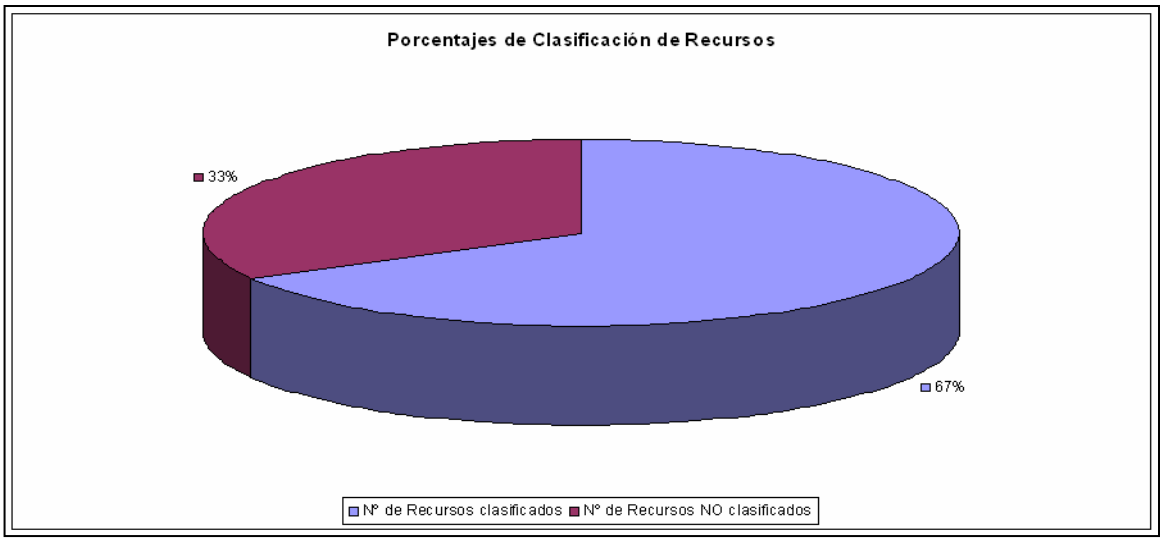


Figura 4.56. Porcentajes de Clasificación de Recursos de la Prueba 18.

La figura 4.56 nos indica que, en esta prueba, se ha conseguido un porcentaje de clasificación aceptable de un 67% de los recursos totales.

4.2.2.1.19. Prueba 19.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 100.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.5
- *similaritiesSRRThres*: 0.4
- *classifierT*: delta

Una vez más, lo más destacable de los tiempos de ejecución de la pruebas (figura 4.57) es el gran tiempo dedicado a la ejecución del módulo de cálculo de las similaridades recurso-recurso.

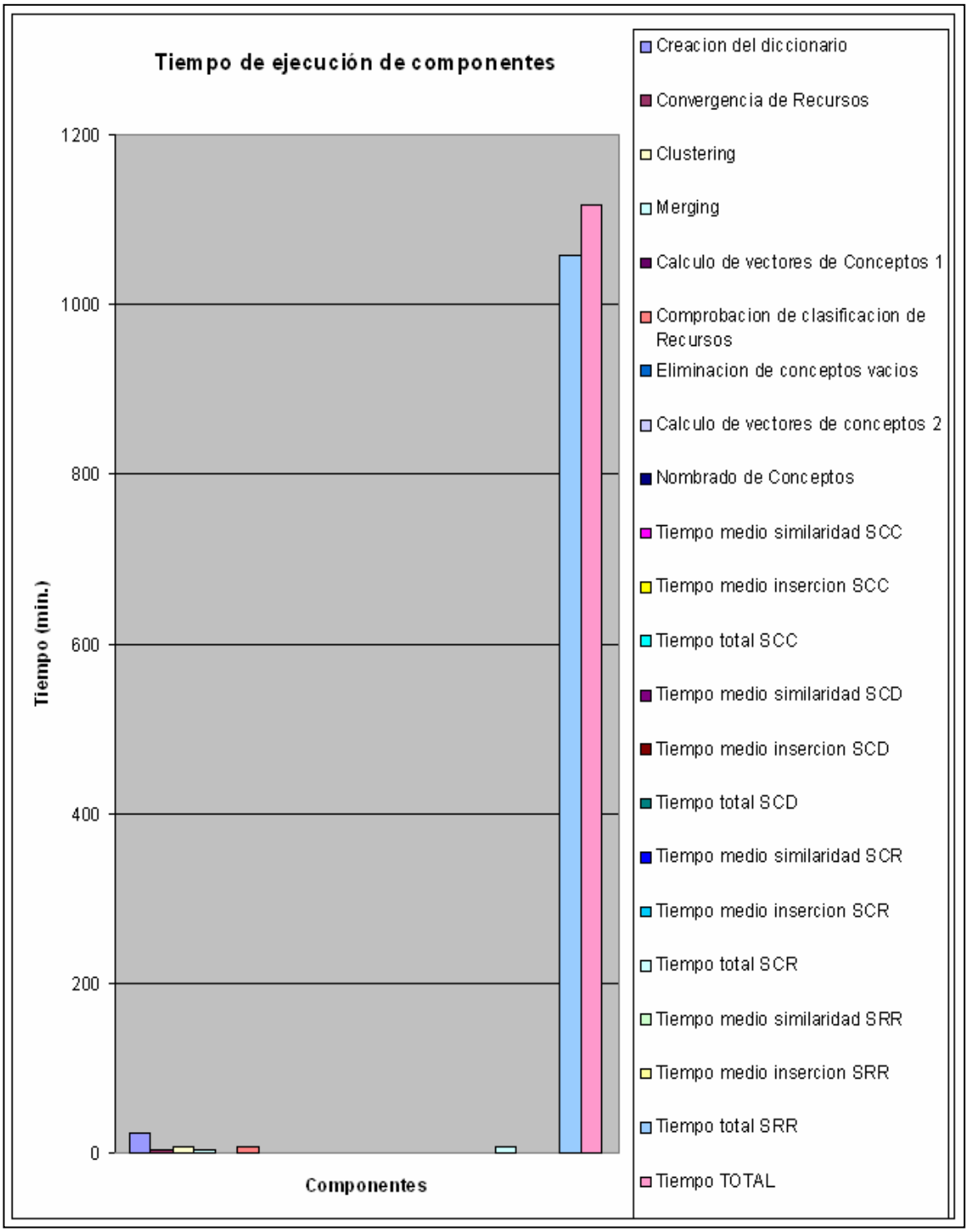


Figura 4.57. Tiempo de Ejecución de Componentes de la Prueba 19.

En la figura 4.58, puede verse que el concepto más destacado por el número de recursos que tiene clasificados es "*ultrasound & baby*".

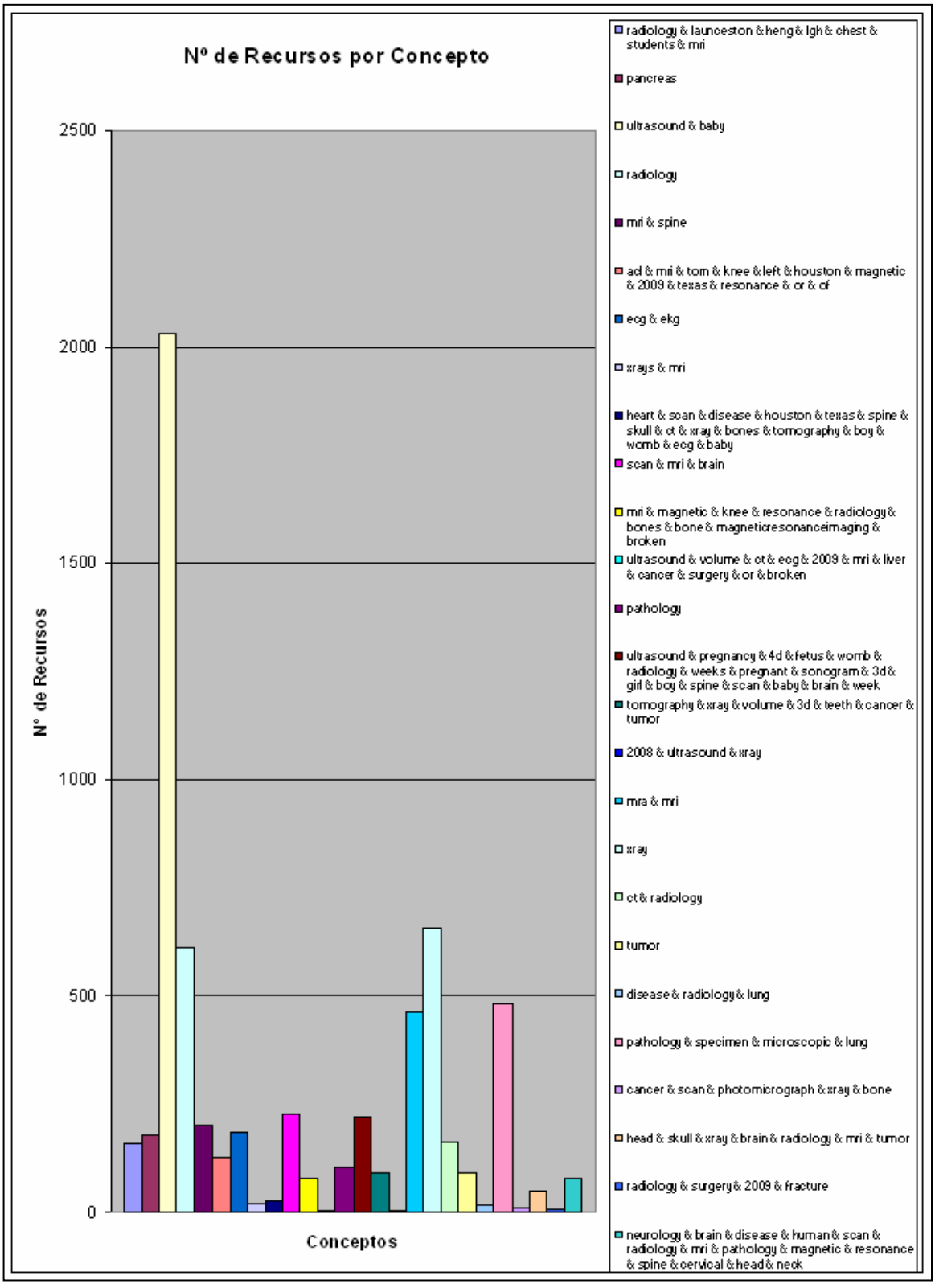


Figura 4.58. Número de Recursos por Concepto de la Prueba 19.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 26.
- Nº de Recursos clasificados: 6282.
- Nº de Recursos no Clasificados: 2297.

- Nº de Recursos *Pending*: 173.
- Nº de Recursos *Converged*: 2124.

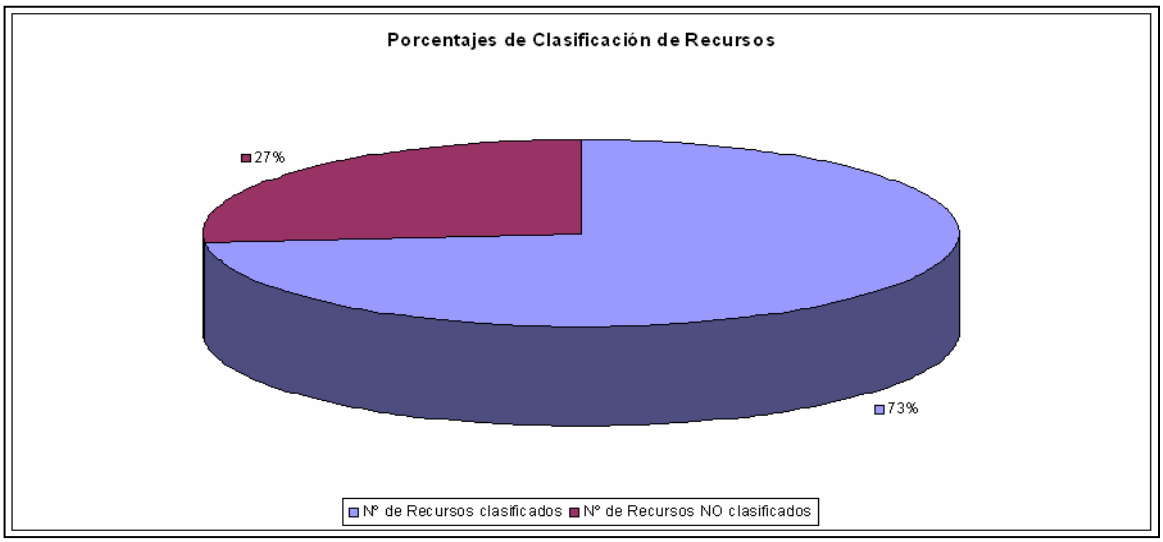


Figura 4.59. Porcentajes de Clasificación de Recursos de la Prueba 19.

Los datos mostrados en la figura 4.59 indican que se ha conseguido un porcentaje de recursos clasificados bastante bueno, de un 73 %.

4.2.2.1.20. Prueba 20.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.75
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim

Como puede apreciarse en la figura 4.60, una vez más lo más destacable en los tiempos de ejecución de los componentes es la amplia ocupación del tiempo de ejecución del cálculo de las similaridades recurso-recurso.

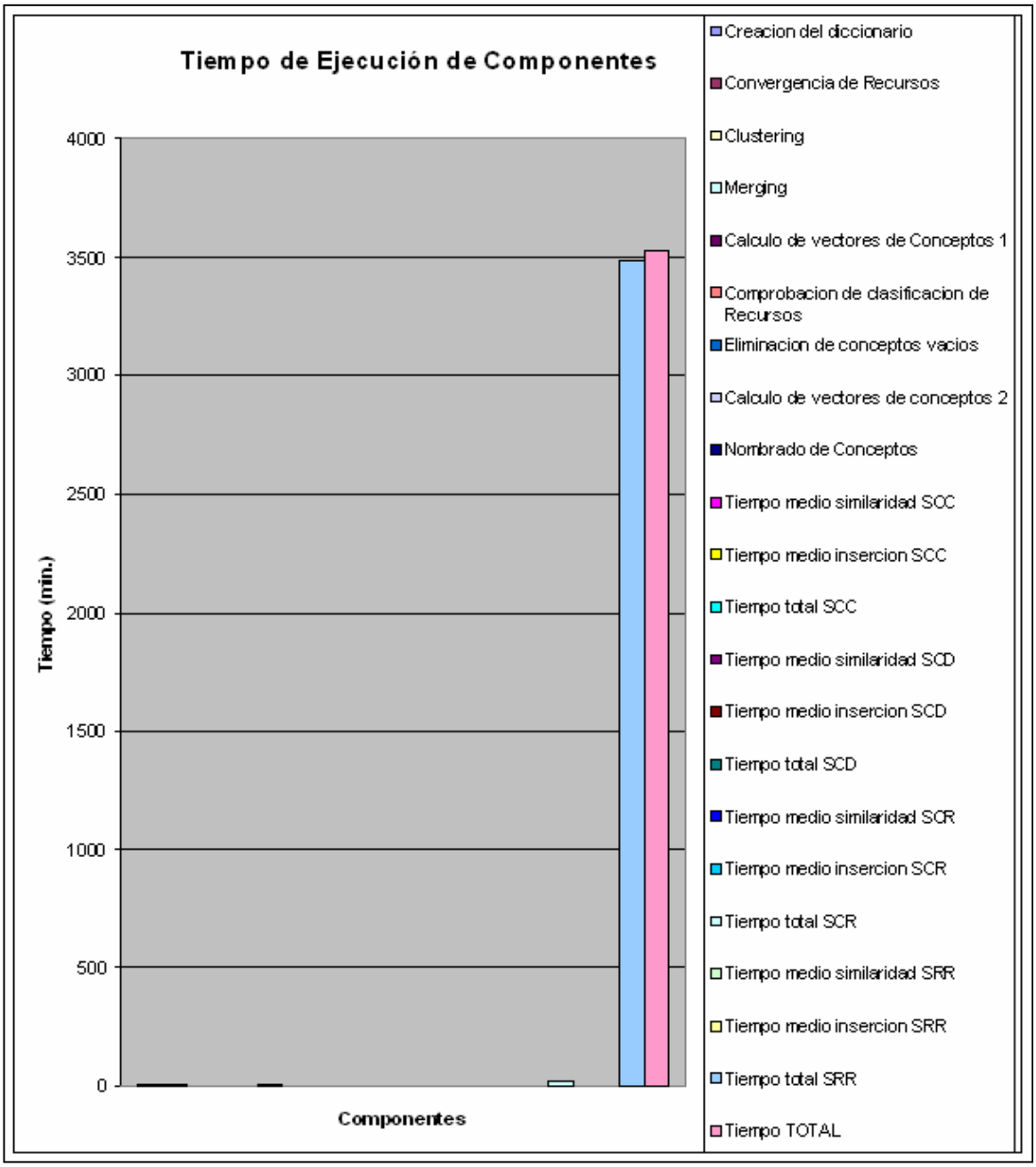


Figura 4.60. Tiempo de Ejecución de los Componentes de la Prueba 20.

En la figura 4.61 puede verse que los recursos se han repartido una forma relativamente equitativa entre los conceptos *pathology*, *mri*, *radiology*, *xray*, "*baby & ultrasound*" y *ultrasound*.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 15.
- Nº de Recursos clasificados: 7757.
- Nº de Recursos no Clasificados: 822.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 0.

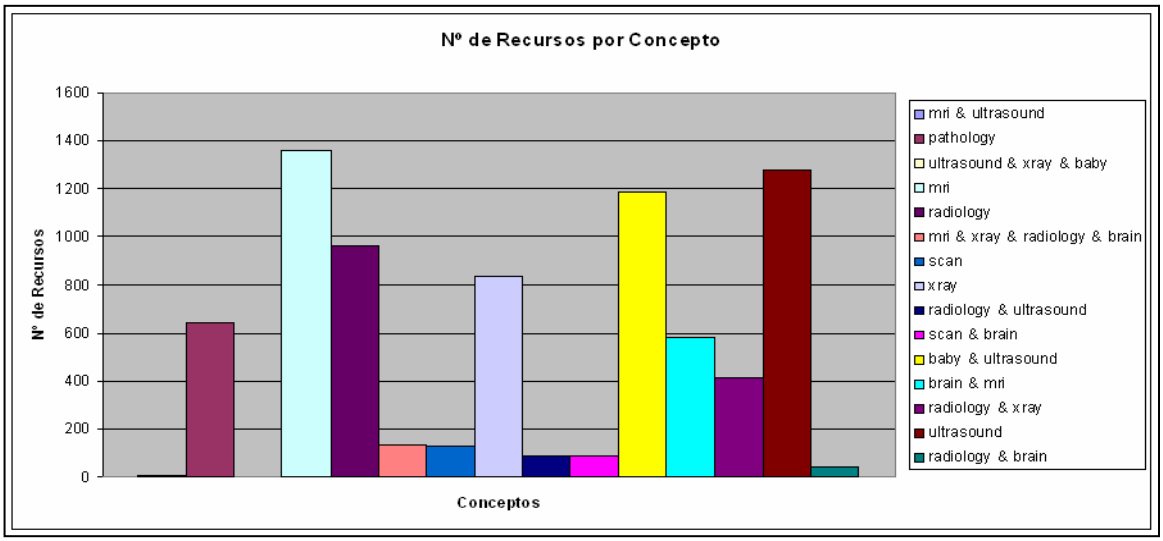


Figura 4.61. Número de Recursos por Concepto de la Prueba 20.

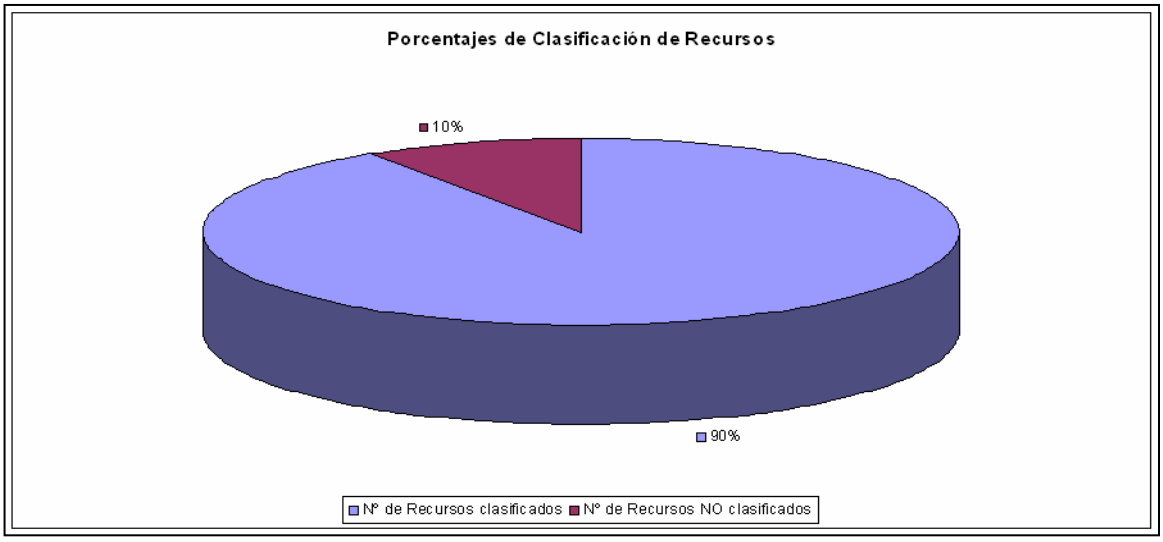


Figura 4.62. Porcentajes de Clasificación de Recursos de la Prueba 20.

Gracias a los datos de clasificación y la figura 4.62, podemos concluir que esta prueba un porcentaje bastante alto de clasificación de recursos, el 90% del total.

4.2.2.1.21. Prueba 21.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.9
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim

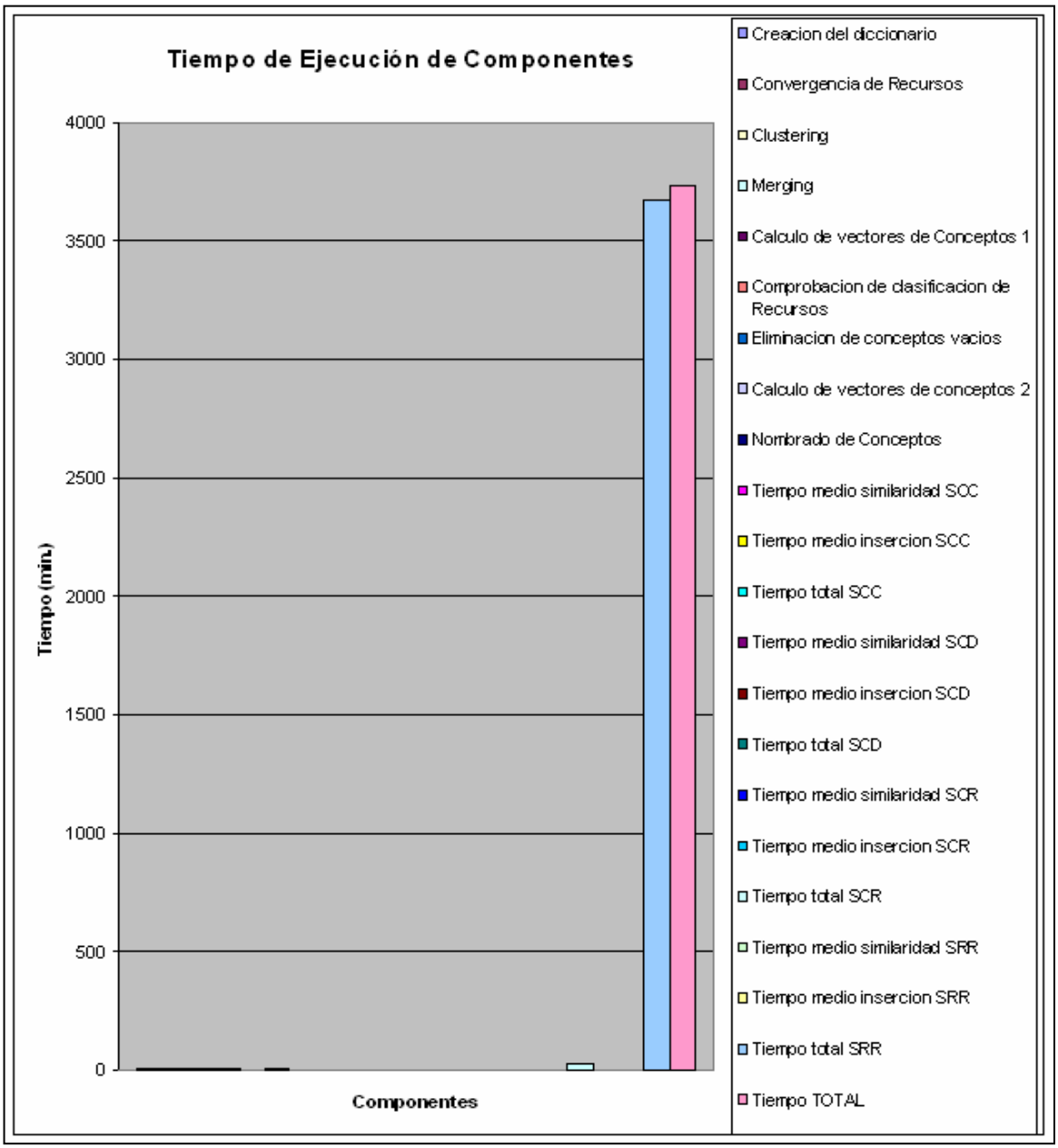


Figura 4.63. Tiempo de Ejecución de los Componentes de la Prueba 21.

Los tiempos de ejecución vuelven a indicar en esta prueba que la mayor parte del tiempo total empleado es para el cálculo de *SRR*.

En la figura 4.64 puede apreciarse que el concepto más poblado es *ultrasound*, seguido de cerca por *mri*.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 19.
- Nº de Recursos clasificados: 7757.
- Nº de Recursos no Clasificados: 822.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 0.

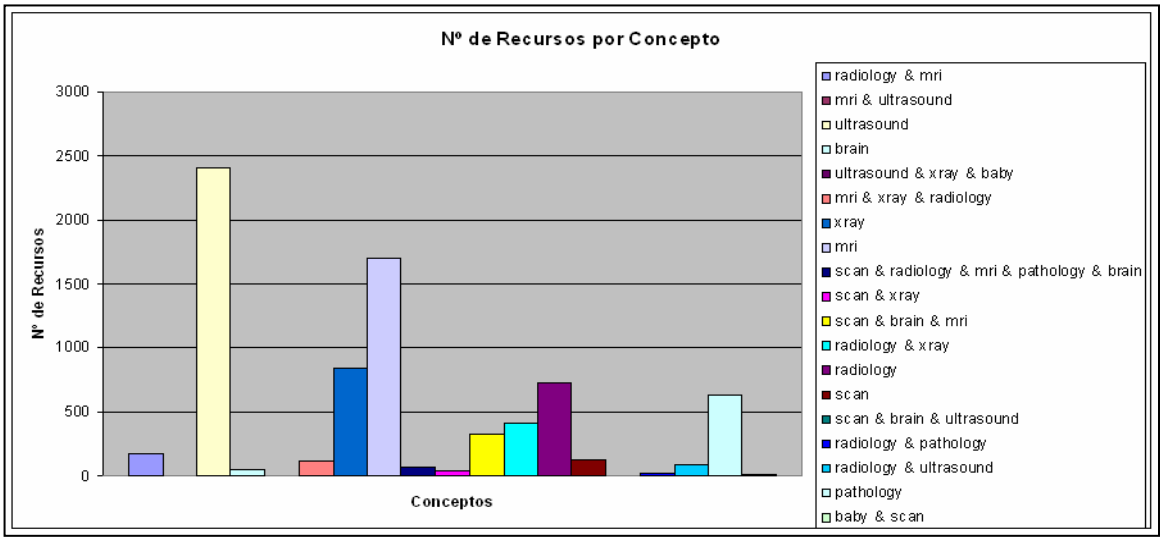


Figura 4.64. Número de Recursos por Concepto de la Prueba 21.

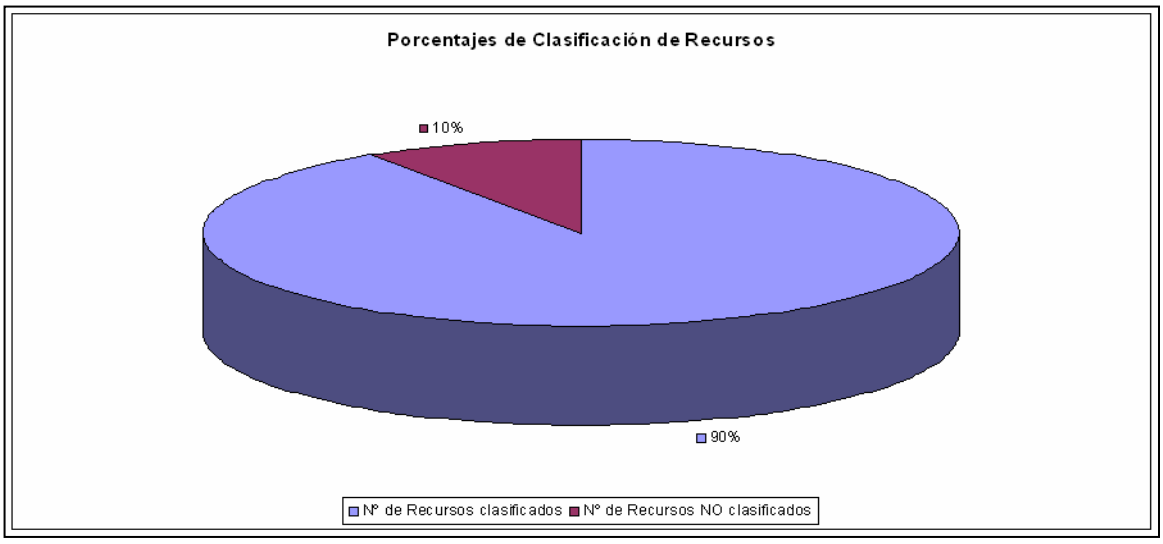


Figura 4.65. Porcentajes de Clasificación de Recursos de la Prueba 21.

En la realización de esta prueba se ha conseguido un porcentaje bastante elevado de clasificación de recursos, un 90% de los recursos.

4.2.2.1.22. Prueba 22.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.3
- *similaritiesMinv*: 0.1

- *similaritiesSRRThres*: 0.4
- *classifierT*: sim

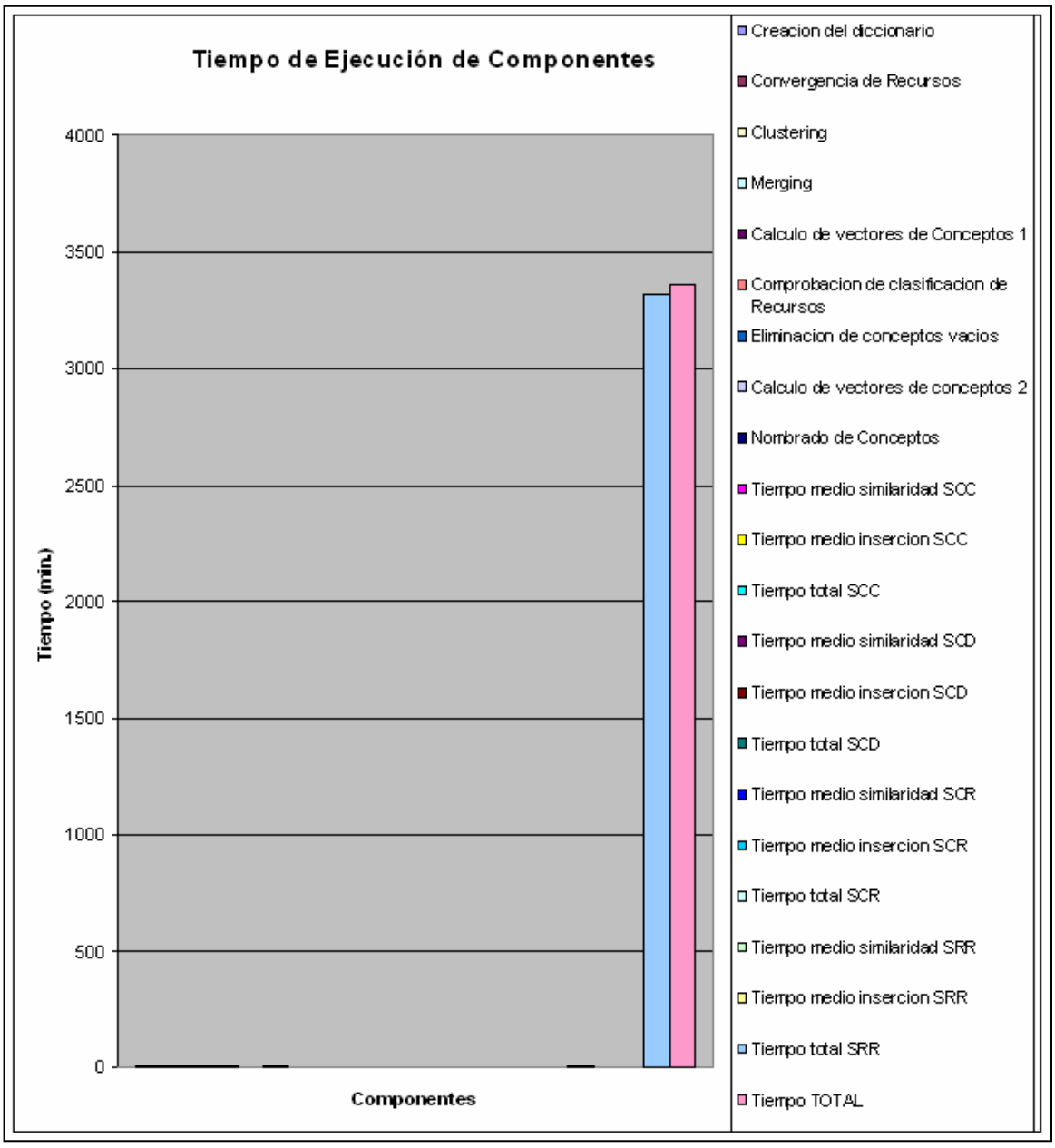


Figura 4.66. Tiempo de Ejecución de los Componentes de la Prueba 22.

En esta prueba también se emplea la gran mayoría del tiempo de ejecución para el cálculo de *SRR*.

La figura 4.67 indica que en esta prueba tan sólo se han generado seis conceptos. Aquellos que destacan por su mayor número de recursos asociados son: *ultrasound* y "*mri & xray*".

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 6.
- Nº de Recursos clasificados: 7757.
- Nº de Recursos no Clasificados: 822.

- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 0.

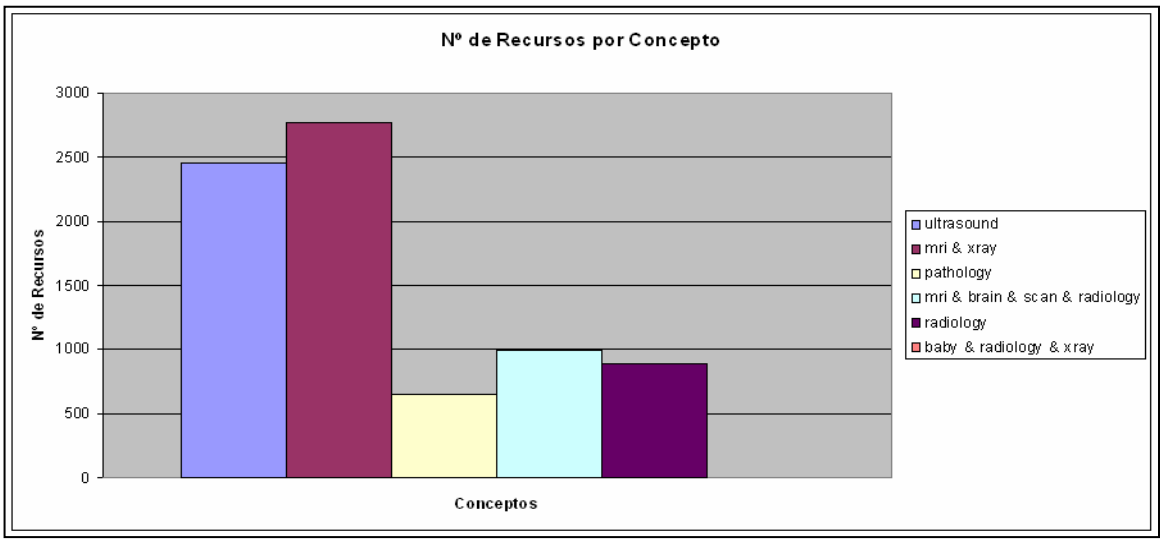


Figura 4.67. Número de Recursos por Concepto de la Prueba 22.

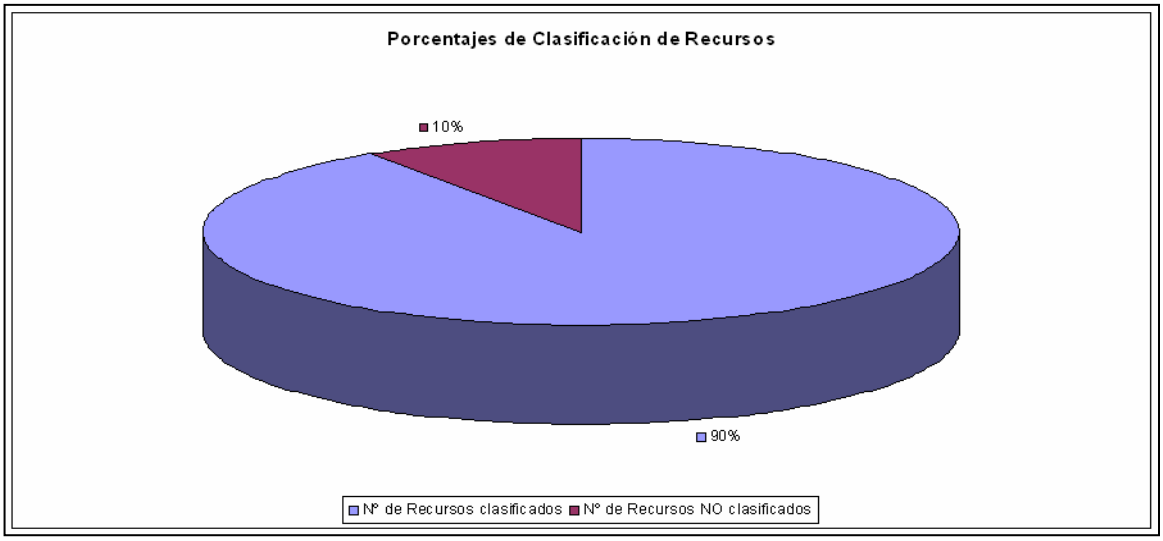


Figura 4.68. Porcentajes de Clasificación de Recursos de la Prueba 22.

Se ha conseguido en esta prueba un alto porcentaje de recursos clasificados, el 90% de ellos (figura 4.68), a pesar de que se hayan generado solamente seis conceptos.

4.2.2.1.23. Prueba 23.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.5

- *similaritiesMinv*: 0.1
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim

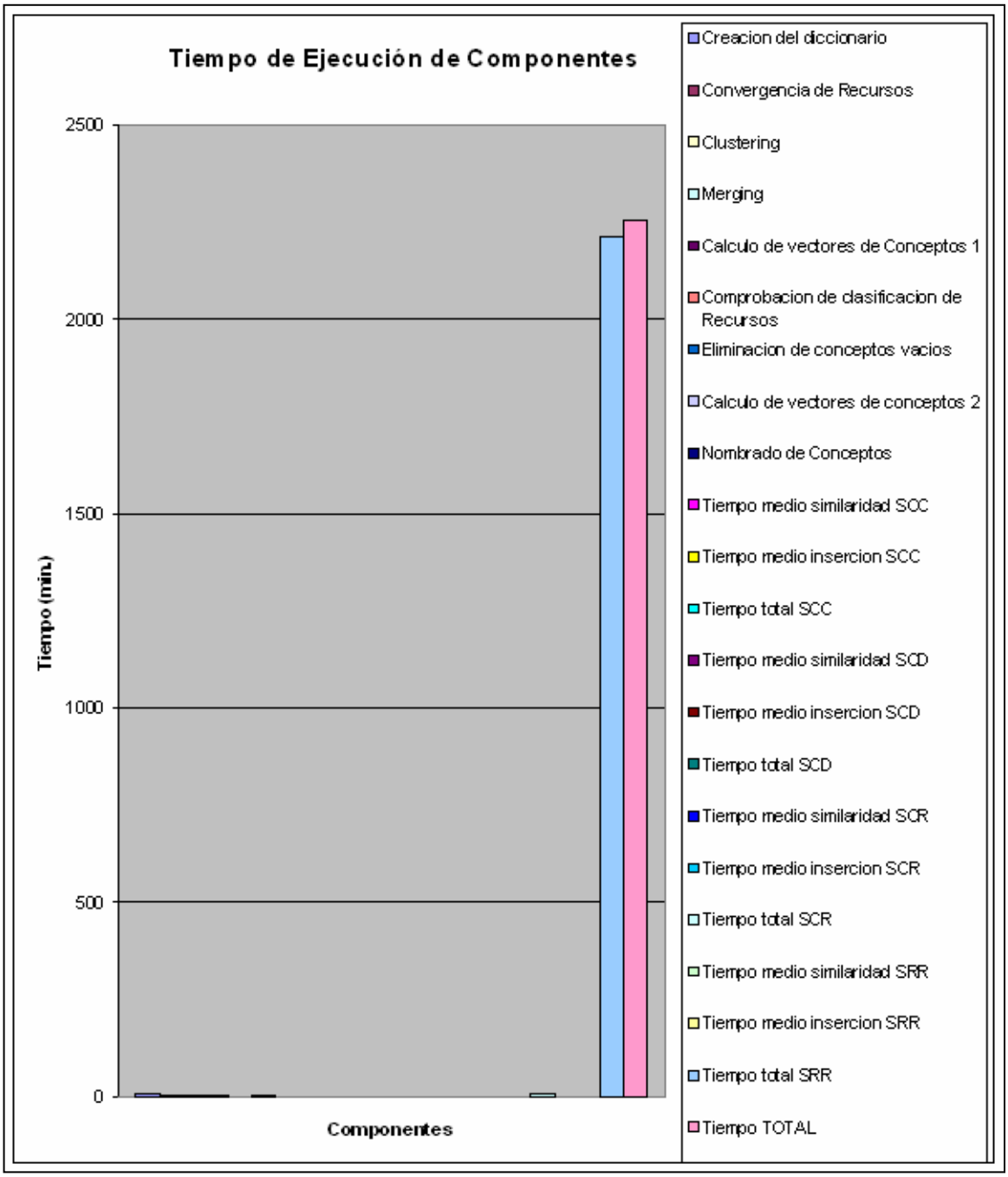


Figura 4.69. Tiempo de Ejecución de los Componentes de la Prueba 23.

Una vez más, la mayor parte del tiempo de ejecución de la prueba ha sido empleado para el cálculo de las similaridades recurso-recurso.

En esta prueba, tal y como se muestra en la figura 4.70, se han creado pocos conceptos, concentrando la mayoría de las clasificaciones en tres de ellos: *mri*, "*ultrasound & xray*" y *radiology*.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 7.
- Nº de Recursos clasificados: 6572.
- Nº de Recursos no Clasificados: 2007.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 1185.

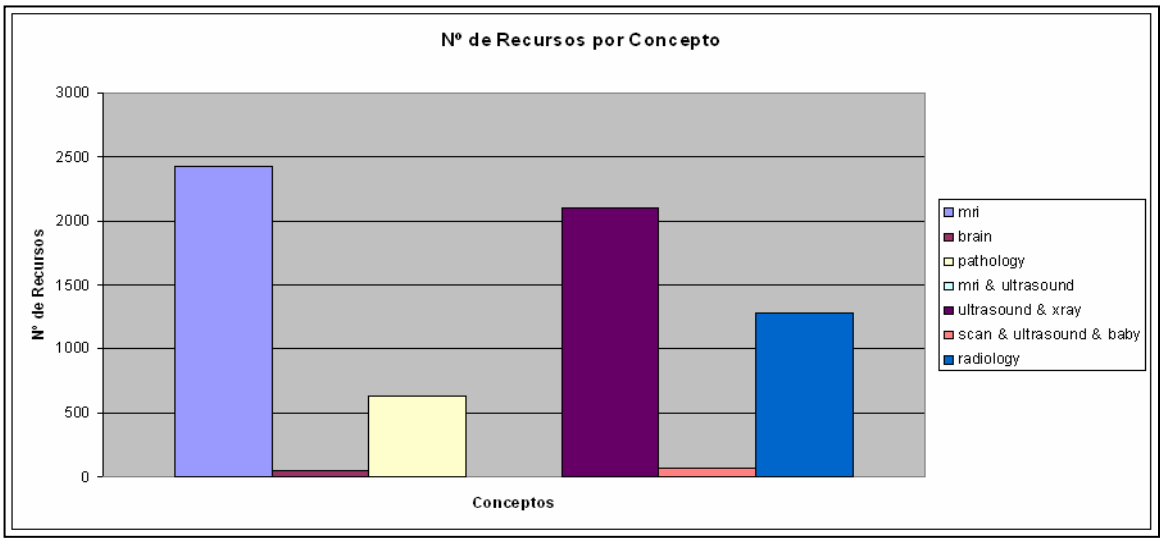


Figura 4.70. Número de Recursos por Concepto de la Prueba 23.

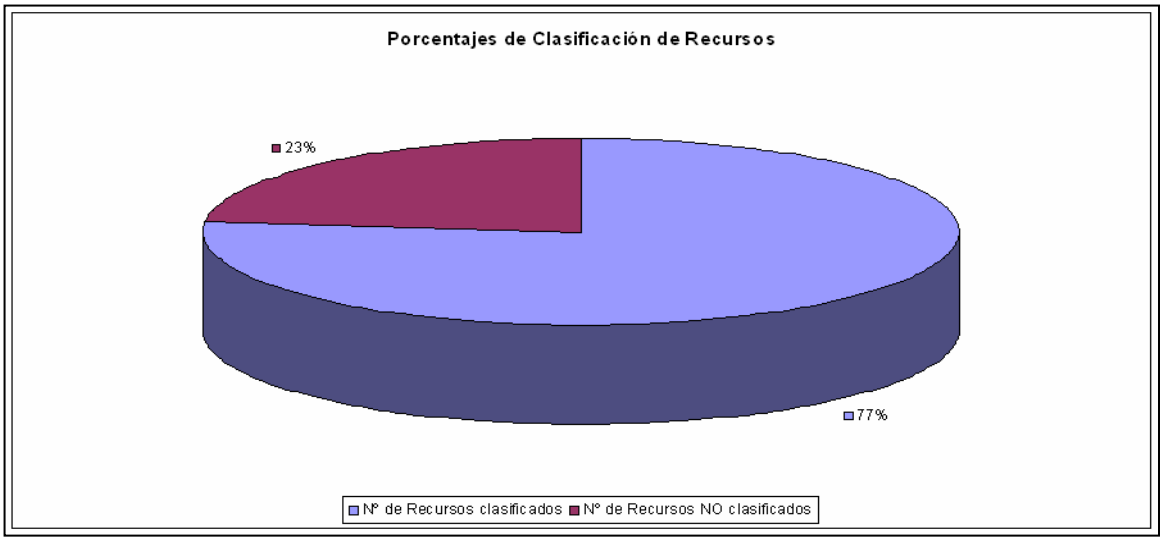


Figura 4.71. Porcentajes de Clasificación de Recursos de la Prueba 23.

En la figura 4.71, puede apreciarse que, aunque habiendo un número pequeño de recursos generados, se ha conseguido un porcentaje relativamente elevado de clasificación de recursos, en torno al 77% del total de los recursos.

4.2.2.1.24. Prueba 24.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.2
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim

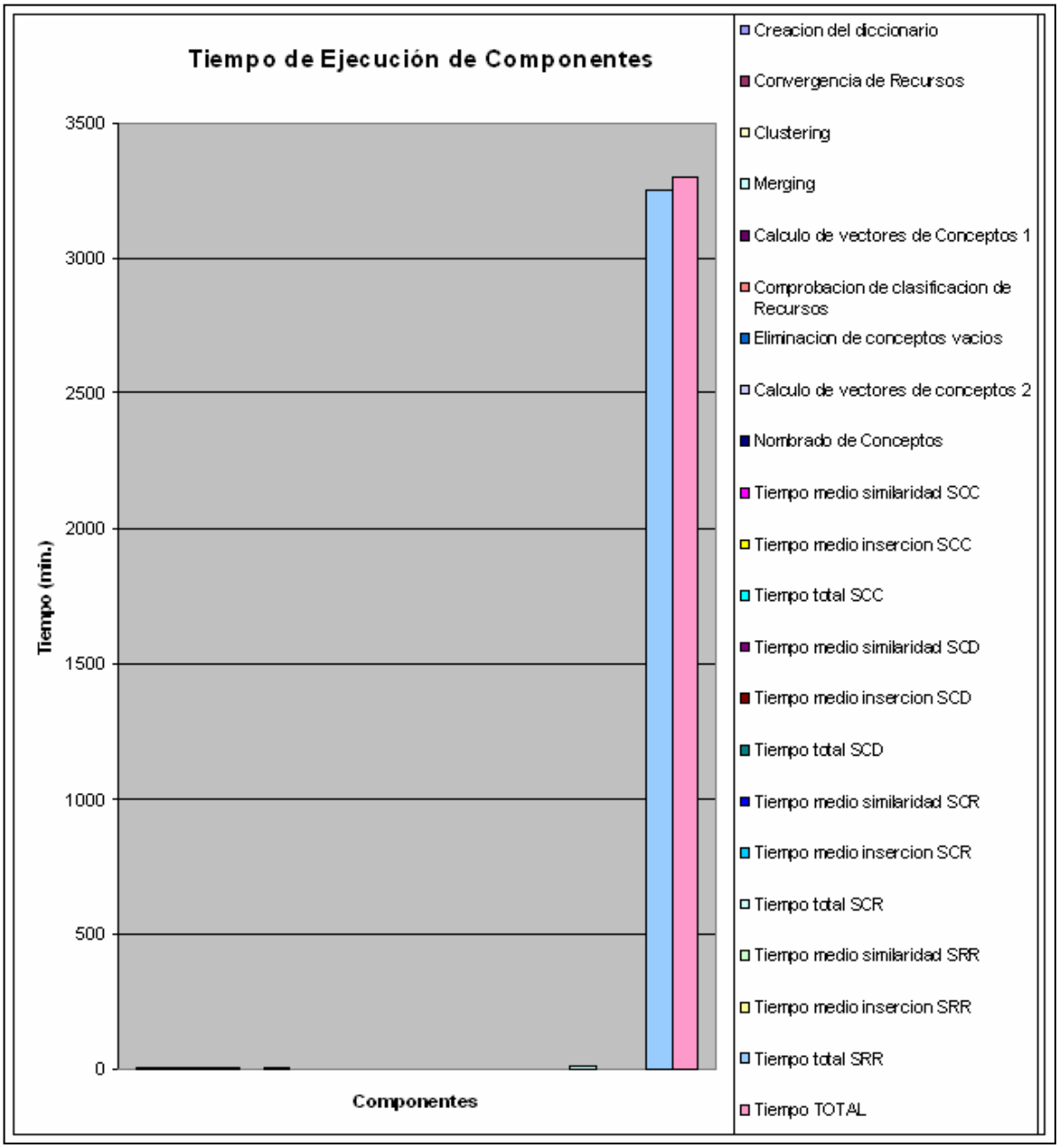


Figura 4.72. Tiempo de Ejecución de los Componentes de la Prueba 24.

En esta prueba también se ha empleado la mayor parte del tiempo de ejecución para el cálculo de *SRR*.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 8.
- Nº de Recursos clasificados: 7757.
- Nº de Recursos no Clasificados: 822.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 0.

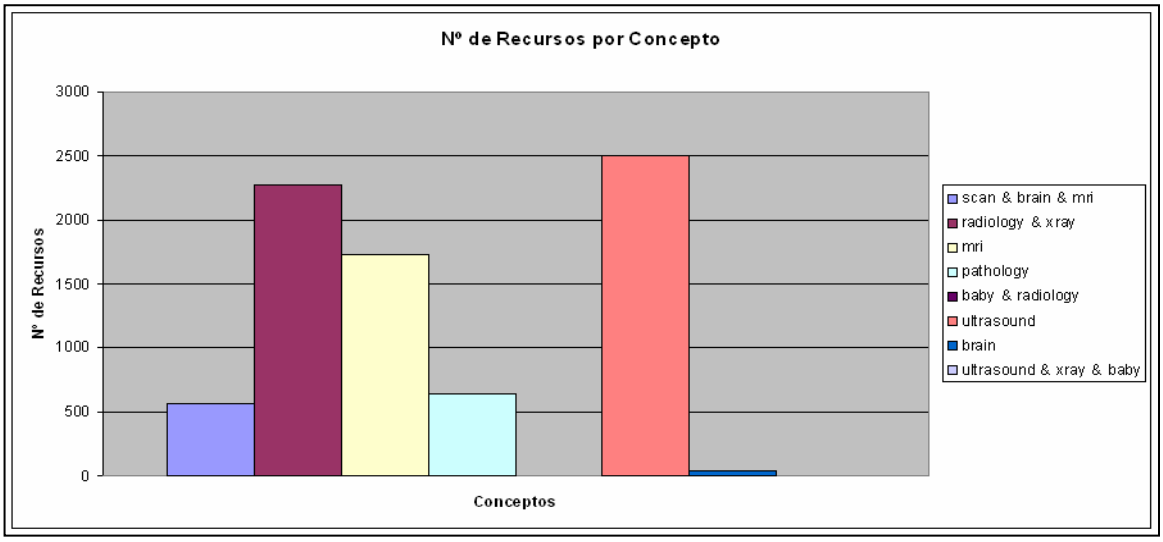


Figura 4.73. Número de Recursos por Concepto de la Prueba 24.

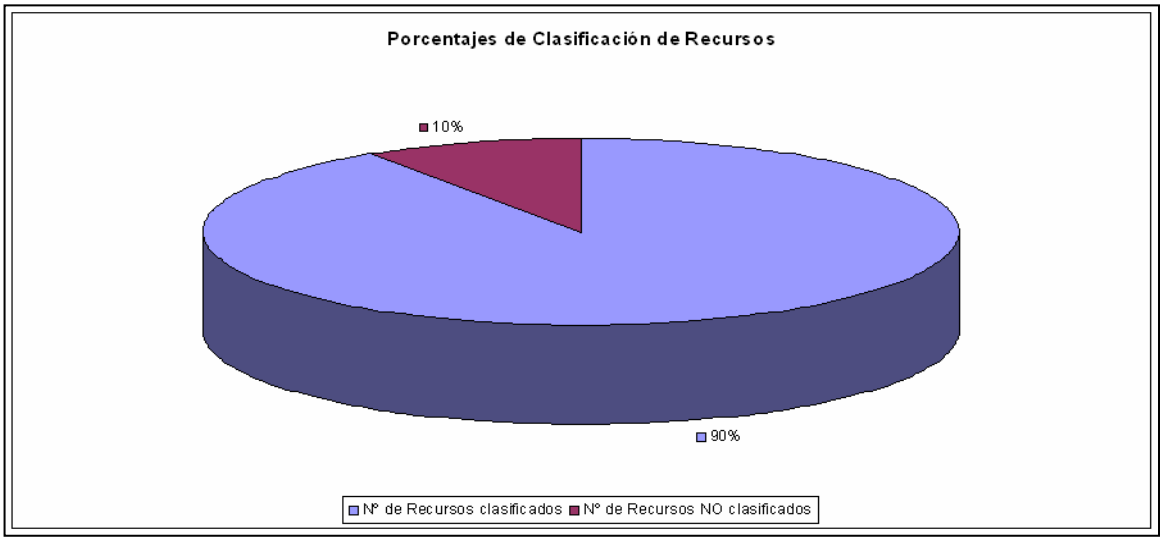


Figura 4.74. Porcentajes de Clasificación de Recursos de la Prueba 24.

Con los datos anteriormente mostrados, puede indicarse que en esta prueba se ha conseguido un índice bastante alto de clasificación de recursos (el 90%) a pesar del bajo número de conceptos generados (8 conceptos). Siendo entre estos los de mayor población de recursos clasificados en ellos: *ultrasound*, "*radiology & xray*" y *mri*.

4.2.2.1.25. Prueba 25.

La configuración de los parámetros de esta prueba son los siguientes:

- *dictionaryMina*: 500.
- *convergenceMina*: 1
- *mergingThreshold*: 0.5
- *classifierThreshold*: 0.1
- *similaritiesMinv*: 0.5
- *similaritiesSRRThres*: 0.4
- *classifierT*: sim

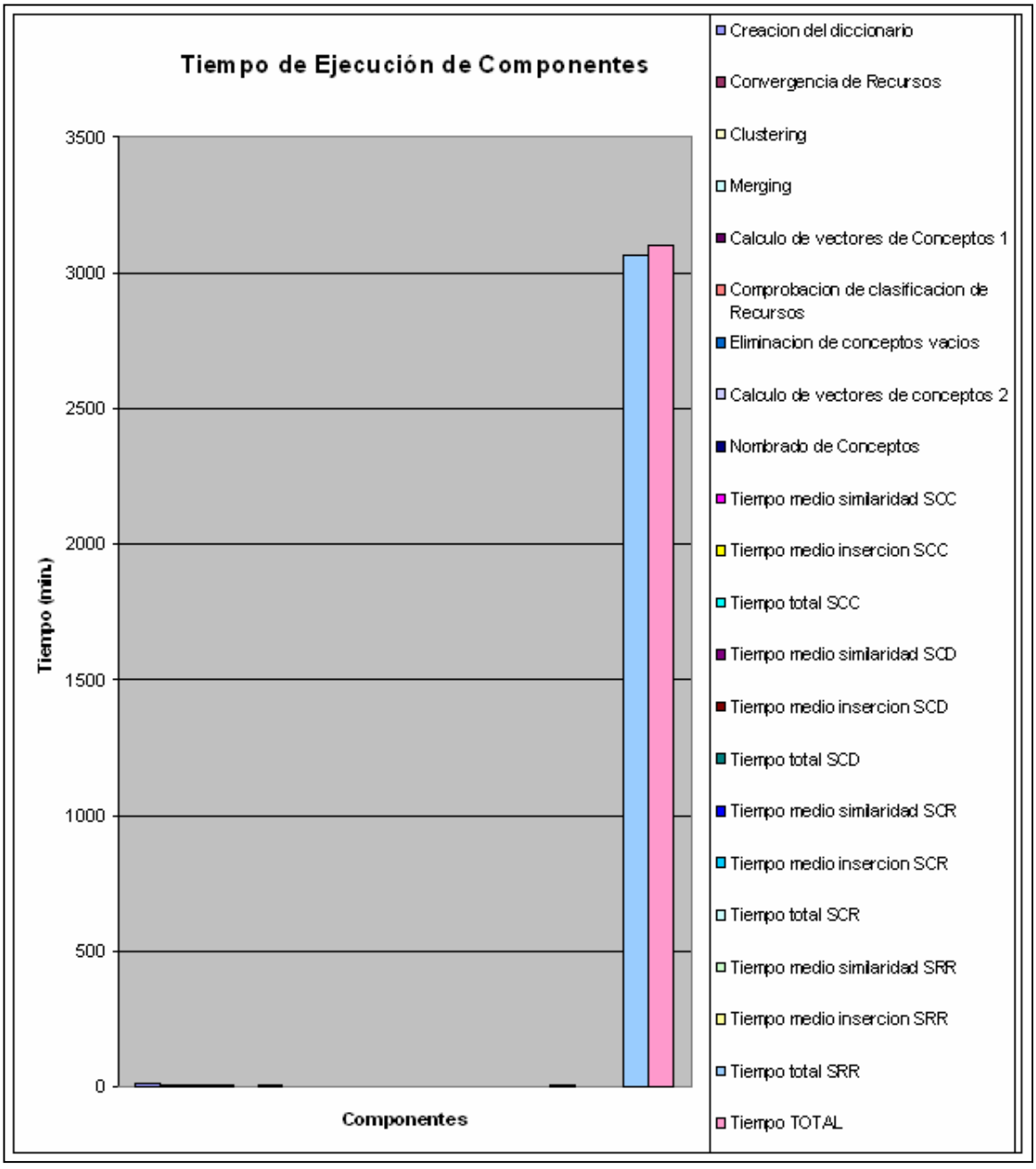


Figura 4.75. Tiempo de Ejecución de los Componentes de la Prueba 25.

En esta última prueba, tal y como indica la figura 4.74, también se ha empleado la mayoría del tiempo de ejecución de la aplicación para el cálculo de las similaridades recurso-recurso.

En esta prueba también se ha generado un número reducido de conceptos, seis en total. Destacan por el número de recursos clasificados en ellos los conceptos: *ultrasound*, *mri*, y *xray*.

Los datos sobre la clasificación de los recursos son los siguientes:

- Nº de Conceptos: 6.
- Nº de Recursos clasificados: 7703.
- Nº de Recursos no Clasificados: 876.
- Nº de Recursos *Pending*: 822.
- Nº de Recursos *Converged*: 54.

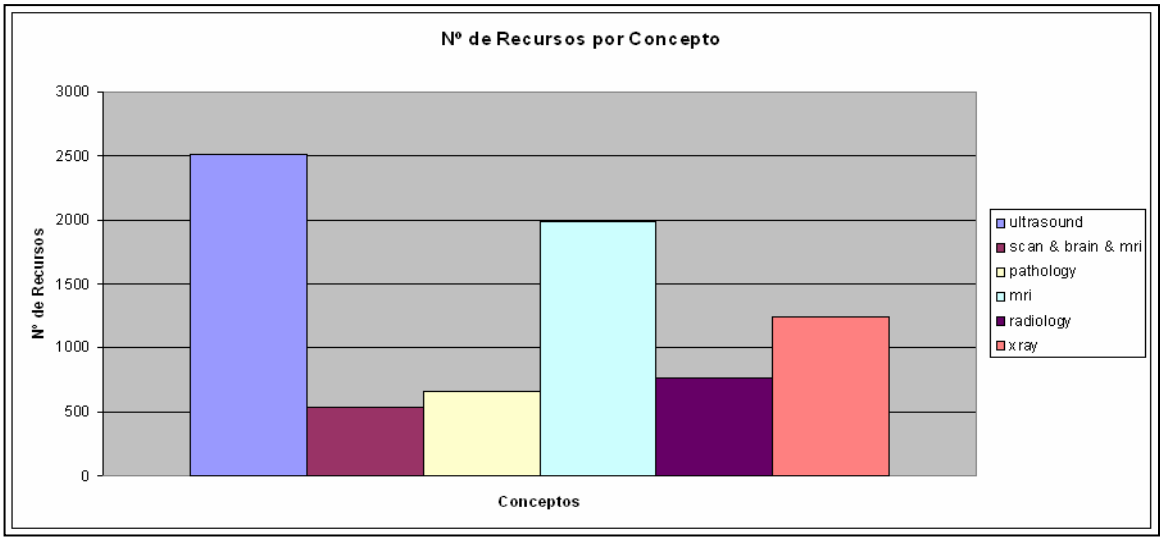


Figura 4.76. Número de Recursos por Concepto de la Prueba 25.

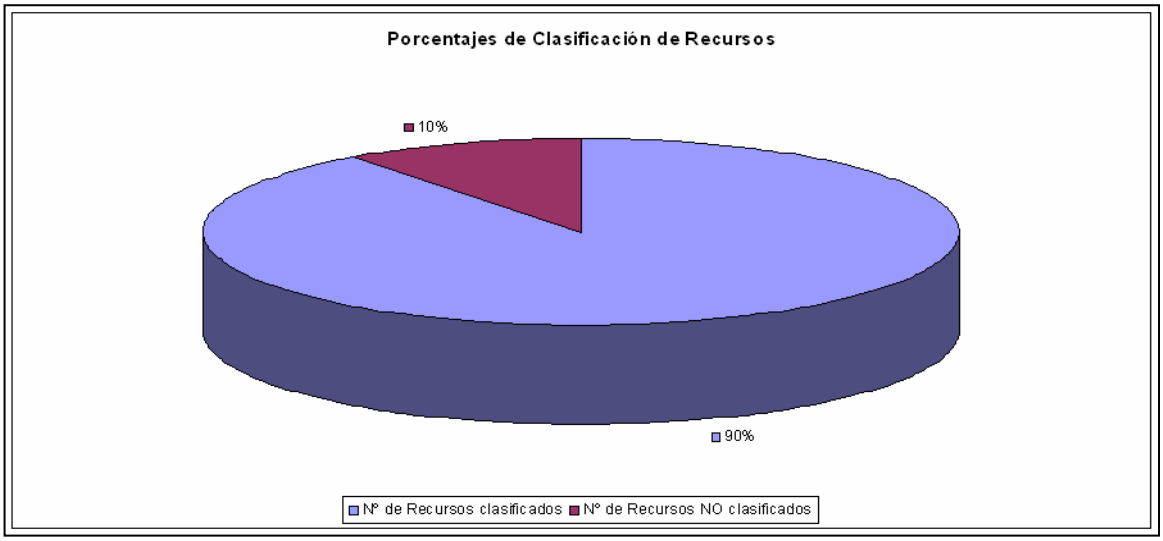


Figura 4.77. Porcentajes de Clasificación de Recursos de la prueba 25.

En esta prueba también se ha conseguido un porcentaje bastante alto de clasificación de los recursos, el 90% de ellos, aun habiéndose generado pocos conceptos.

4.2.3. Análisis de los Resultados.

Para el análisis de los resultados obtenidos en las pruebas se ha optado por dos caminos:

1. Obtener una serie de estadísticas globales de todas las pruebas que comparen los tiempos de ejecución de cada componente, el tiempo total de ejecución y algunos datos sobre la clasificación de los recursos, como el número de conceptos, el número de recursos clasificados y no clasificados…
2. Validar la clasificación de los recursos mediante el uso de *testers*.

4.2.3.1. Estadísticas Globales.

En este apartado van a exponerse de forma conjunta diferentes datos y mediciones extraídos de las pruebas presentadas en el apartado inmediatamente anterior.

4.2.3.1.1. Tiempos de Ejecución de los Componentes de ACoAR.

En esta sección se van a comparar los diferentes tiempos de ejecución de cada componente en cada una de las pruebas.

4.2.3.1.1.1. Tiempos de Creación del Diccionario.

La creación del diccionario se encarga de decidir que etiquetas pertenecen a este, y en generar los vectores representativos de los recursos y de las etiquetas del diccionario.

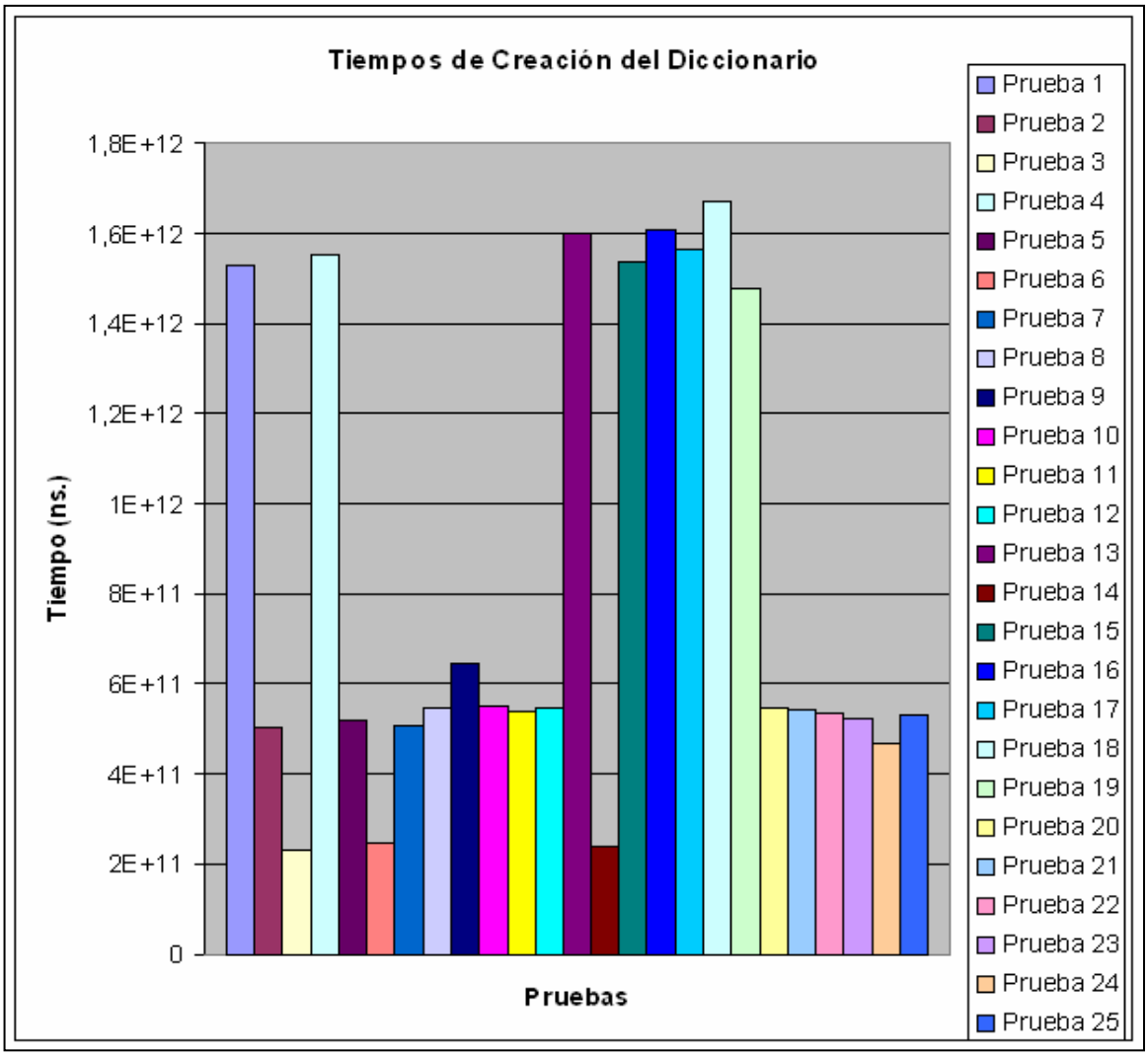


Figura 4.78. Tiempos de Creación del Diccionario de la Prueba 1 a la 25.

En la figura 4.78, puede observarse que aquellas pruebas con un tiempo de creación del diccionario más alto (pruebas 1, 4, 13, 15-19) corresponde a aquellas con el parámetro *dictionaryMina* configurado a 100 anotaciones, y aquellos tiempos más bajos (pruebas 3, 6, 14) corresponden a pruebas con dicho parámetro configurado a 2564. Todo esto es lógico ya que cuanto menor sea el valor asignado a *dictionaryMina*, más etiquetas van a formar parte del diccionario, más dimensiones van a tener los vectores y, por tanto, más tiempo va a llevar el crearlos.

4.2.3.1.1.2. Tiempos de Convergencia de Recursos.

Este componente se encarga de definir que recursos del sistema han convergido y cuales están pendientes de ello.

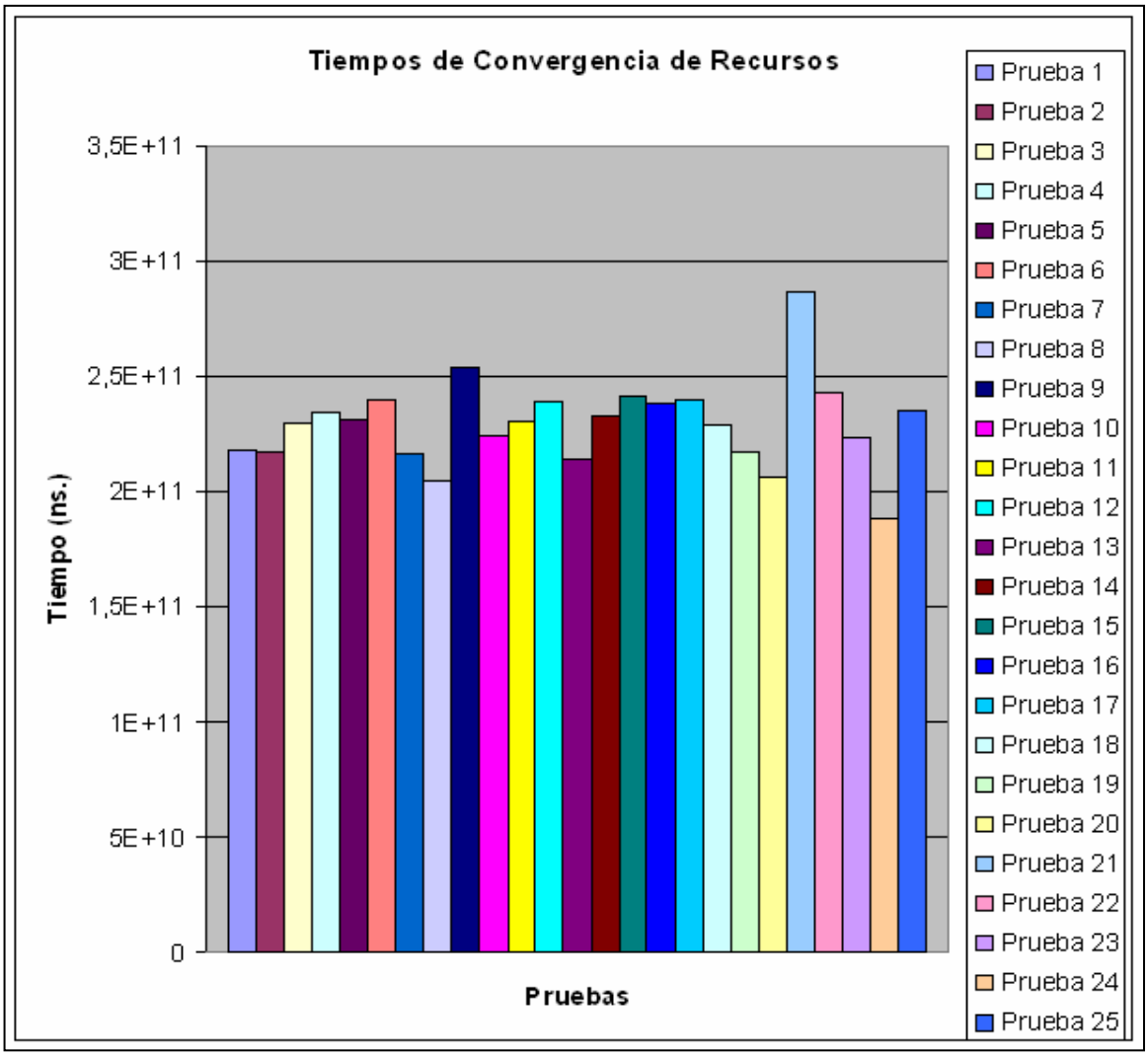


Figura 4.79. Tiempos de Convergencia de Recursos de la Prueba 1 a la 25.

Puede apreciarse en la figura 4.79, que el tiempo de convergencia de los recursos es bastante similar en todas las pruebas. Esto es debido a que, en todas las pruebas, el parámetro *convergenceMina* siempre está configurado a 1, y todas tienen los mismos recursos de entrada.

4.2.3.1.1.3. Tiempos de *Clustering*.

El componente de *clustering* se encarga de crear los conceptos y de clasificar los recursos en ellos.

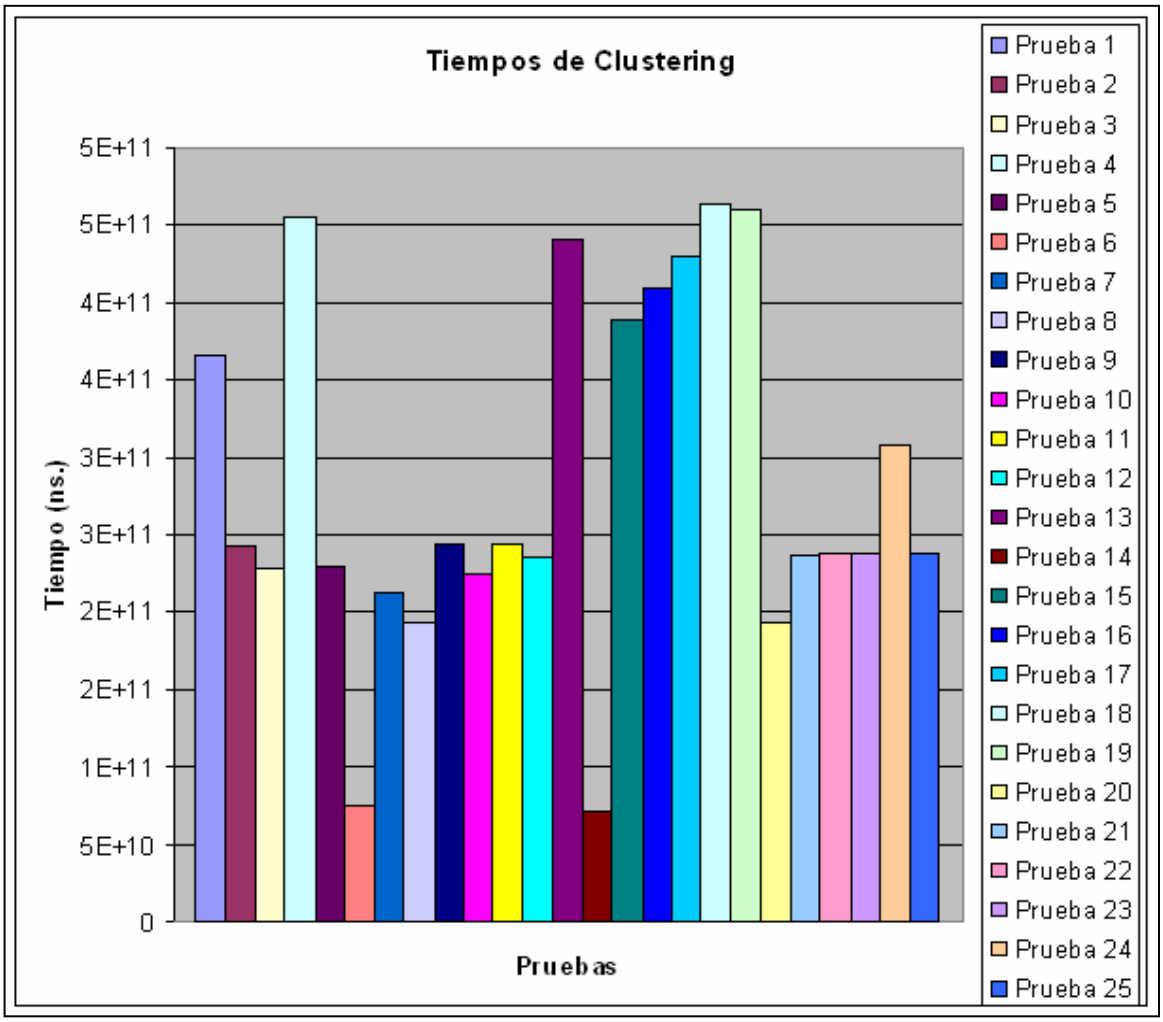


Figura 4.80. Tiempos de Clustering de la Prueba 1 a la 25.

Puede verse en la figura 4.80 que aquellas pruebas con tiempos de *clustering* más elevado coinciden con aquellas que tienen el parámetro *dictionaryMina* configurado a 100 (pruebas 1, 4, 13, 15-19), y aquellas con un tiempo menor coinciden con la configuración de dicho parámetro a 2564 (pruebas 3, 6, 14). Esto es debido a que cuanto menor sea el valor de *dictionaryMina*, más etiquetas estarán en el diccionario, y más dimensiones tendrá el espacio vectorial sobre el que realizar el *clustering*, siendo su cálculo más complejo.

4.2.3.1.1.4. Tiempos de *Merging*.

El componente de *merging* se encarga de unir aquellos conceptos cuya similaridad sea superior a *mergingThreshold*.

Vemos en la figura 4.81 que prácticamente la mayoría de las pruebas ha tenido un tiempo similar, a excepción de las pruebas 6 y 14, con un tiempo significativamente menor, y la prueba 24 con un tiempo mayor. La explicación al fenómeno de las pruebas 6 y 14 puede deberse a que solamente se ha generado un concepto en dichas pruebas, pero entonces también debería ser menor el tiempo de la prueba 3. Además parece que el tiempo de

merging no atiende al número de conceptos, ni al tamaño del diccionario, aquello que a priori podría afectarle más.

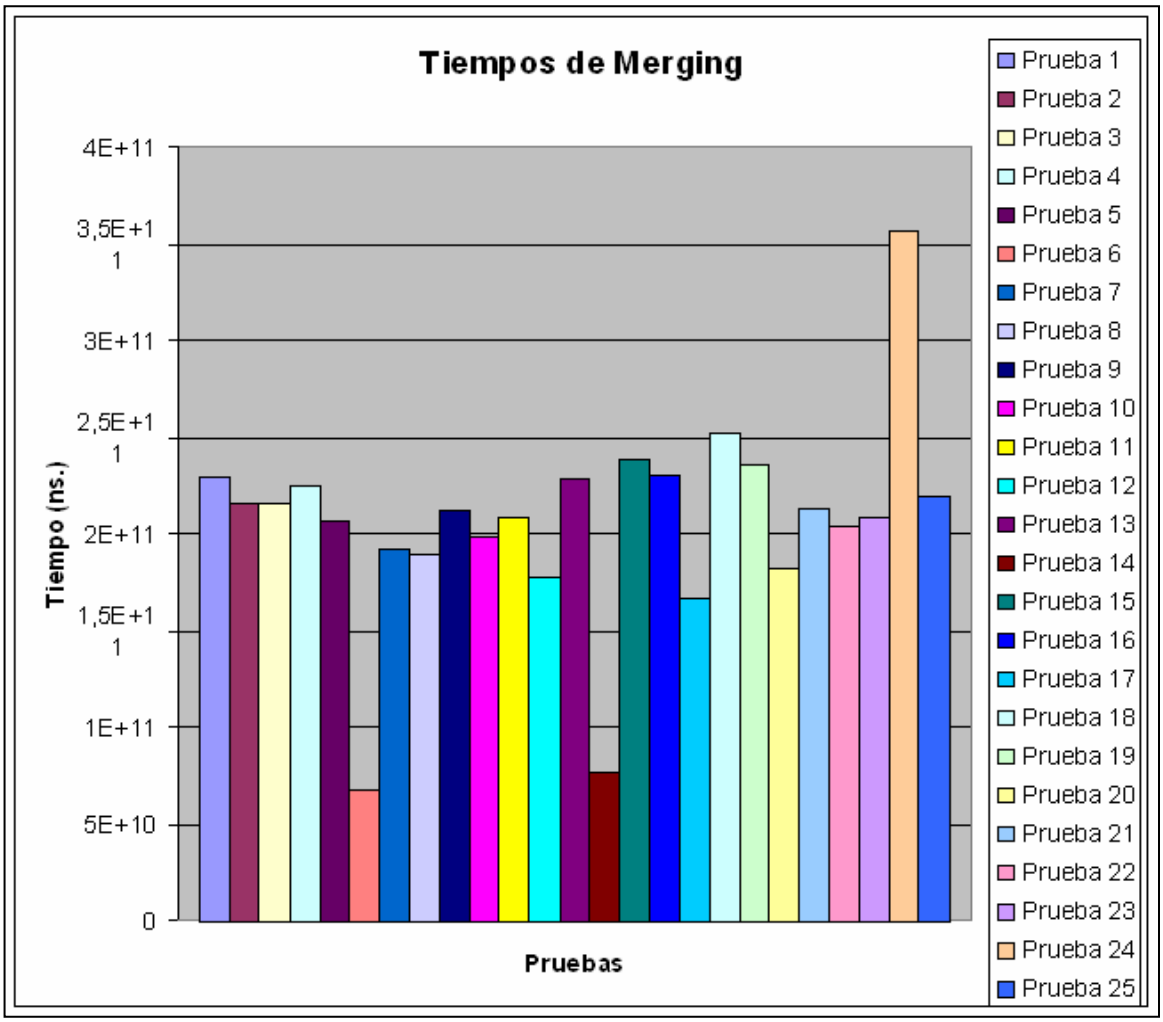


Figura 4.81. Tiempos de Merging de la Prueba 1 a la 25.

4.2.3.1.1.5. Tiempos de Cálculo de Vectores de Conceptos 1.

Esta parte del algoritmo se encarga de recalcular los vectores de los conceptos una vez que se ha realizado la unión de conceptos similares.

Como en otras comparativas anteriores, parece ser que el tiempo para este cálculo de los vectores de los conceptos está directamente relacionado con el número de etiquetas que hay en el diccionario. Al haber más, los vectores tienen más componentes y, por tanto, su recalculo es más costoso.

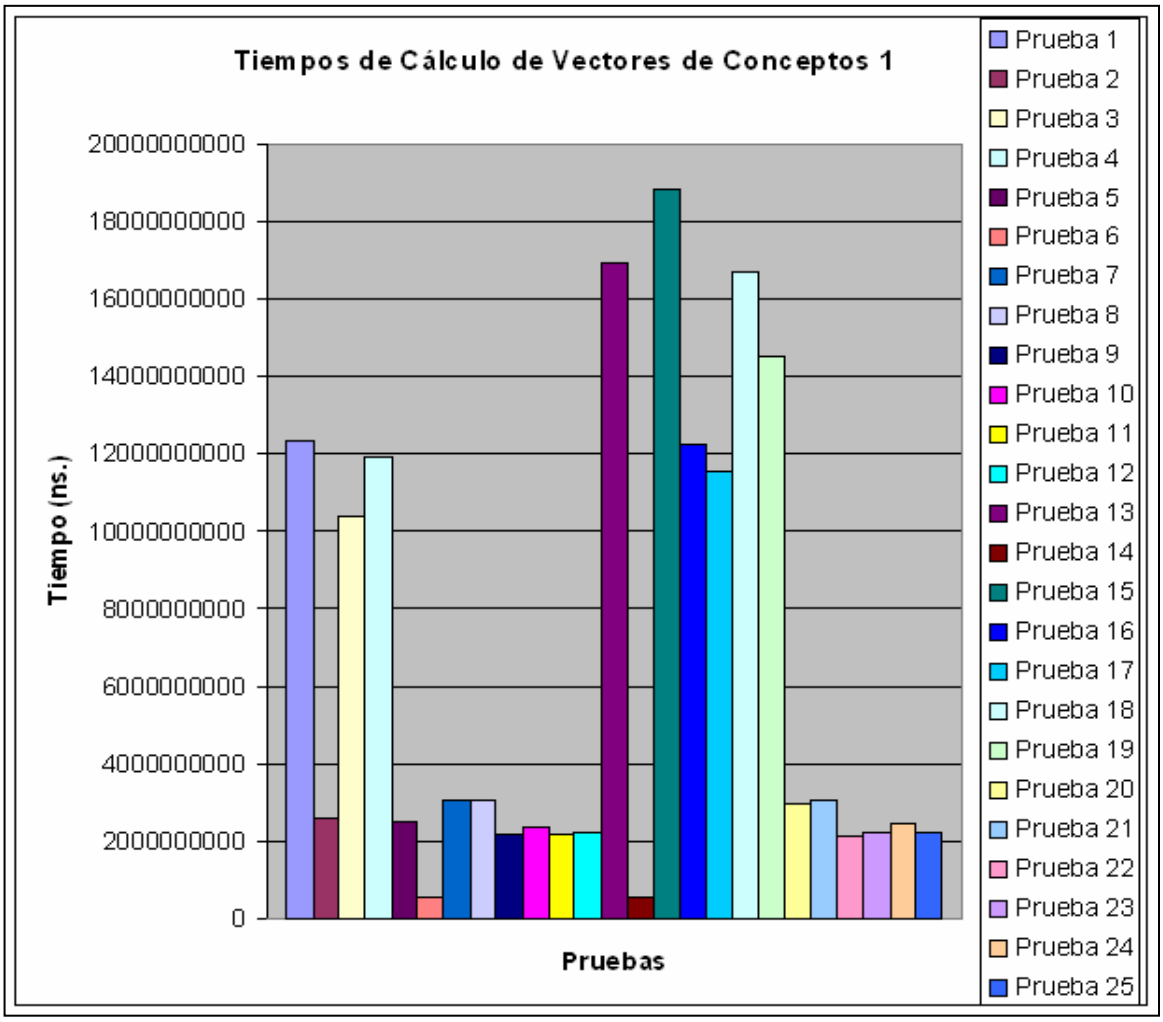


Figura 4.82. Tiempos de Cálculo de Vectores de Conceptos 1 de las Pruebas 1 a 25.

4.2.3.1.1.6. Tiempos de Comprobación de Clasificación de Recursos.

Este componente se encarga de establecer si los recursos están correctamente o incorrectamente clasificados.

En este caso la grafica (figura 4.83), muestra que la mayoría de las pruebas tiene un tiempo de ejecución, de esta parte del algoritmo, similar. Aun así se ven una serie de excepciones: pruebas 3, 6, 9, 10, 14 y 17. Esta serie de pruebas coinciden en que han tenido un porcentaje muy bajo de clasificación. Entonces es lógico que tengan un tiempo de ejecución menor debido a dos posibles situaciones que pueden darse:

- Que al principio del algoritmo no hayan convergido muchos recursos, por tanto, no habría que comprobar la clasificación de muchos recursos.
- O, que en esta parte del algoritmo, se consideren que muchos recursos han sido incorrectamente clasificados y, por tanto, no haya que cambiar su estado en la base de datos a *classified*, ahorrándose el tiempo que ello conlleva.

Entonces, cuantos más recursos se consideren correctamente clasificados, más actualizaciones habrá que hacer en la base de datos y, por tanto, el tiempo de ejecución de este componentes será mayor.

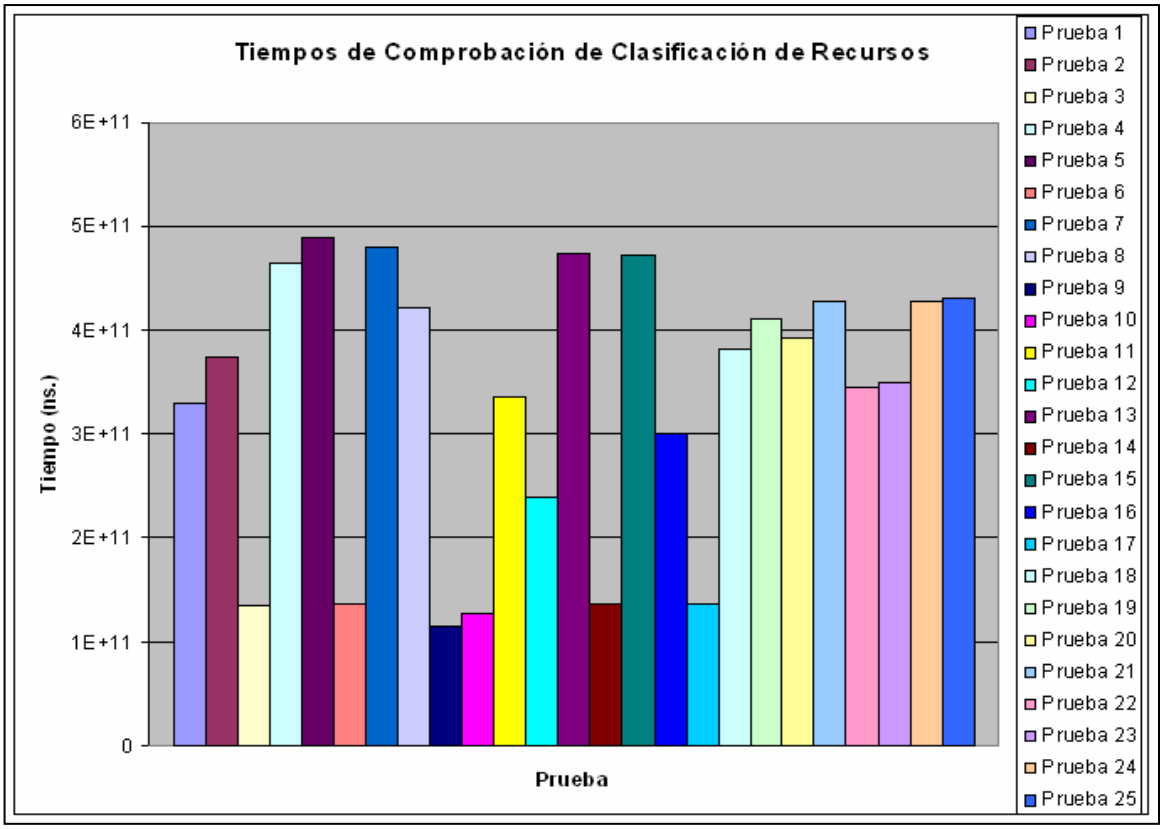


Figura 4.83. Tiempos de Comprobación de Clasificación de Recursos de la Prueba 1 a la 25.

4.2.3.1.1.7. Tiempos de Eliminación de Conceptos Vacíos.

Esta parte del algoritmo se encarga de eliminar del sistema aquellos conceptos que no tienen ningún recurso clasificado en ellos.

Puede verse, en la figura 4.84, que el coste en tiempo es mínimo. Incluso contando la excepción producida en la prueba 17, ya que esta ni llega a un segundo.

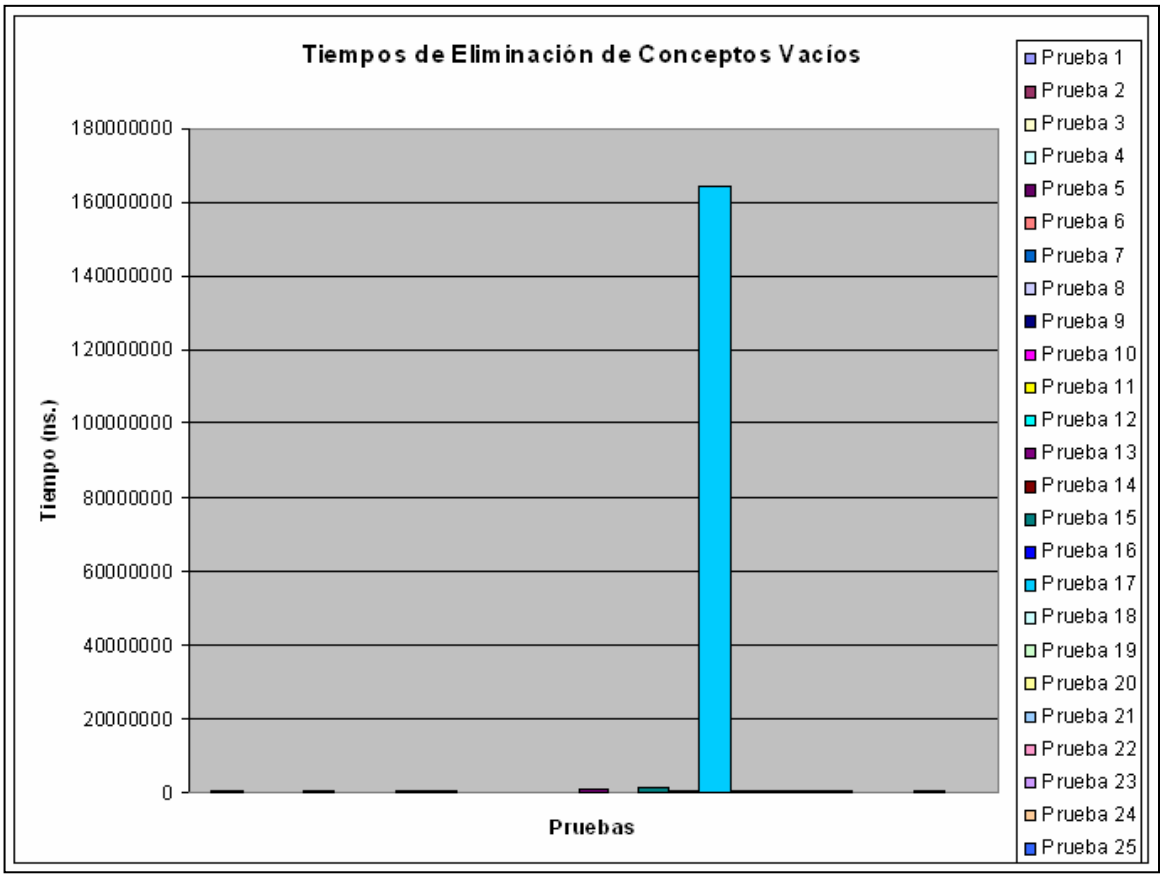


Figura 4.84. Tiempos de Eliminación de Conceptos Vacíos de la Prueba 1 a la 25.

4.2.3.1.1.8. Tiempos de Cálculo de Vectores de Conceptos 2.

Otra vez se vuelven a recalcular los vectores de los conceptos debido a las modificaciones sufridas en los mismos durante la comprobación de la clasificación de los recursos.

Como el anterior caso tratado de cálculo de vectores de los conceptos, parece haber una relación directa entre el tamaño del diccionario de cada prueba con el tiempo que se ha tardado en recalcular los vectores de los conceptos. A excepción de la prueba 17, que está por debajo del tiempo esperado, aunque esto se debe a que dicha prueba tiene un porcentaje muy bajo de clasificación de recursos y, por tanto, la complicación del cálculo se reduce considerablemente.

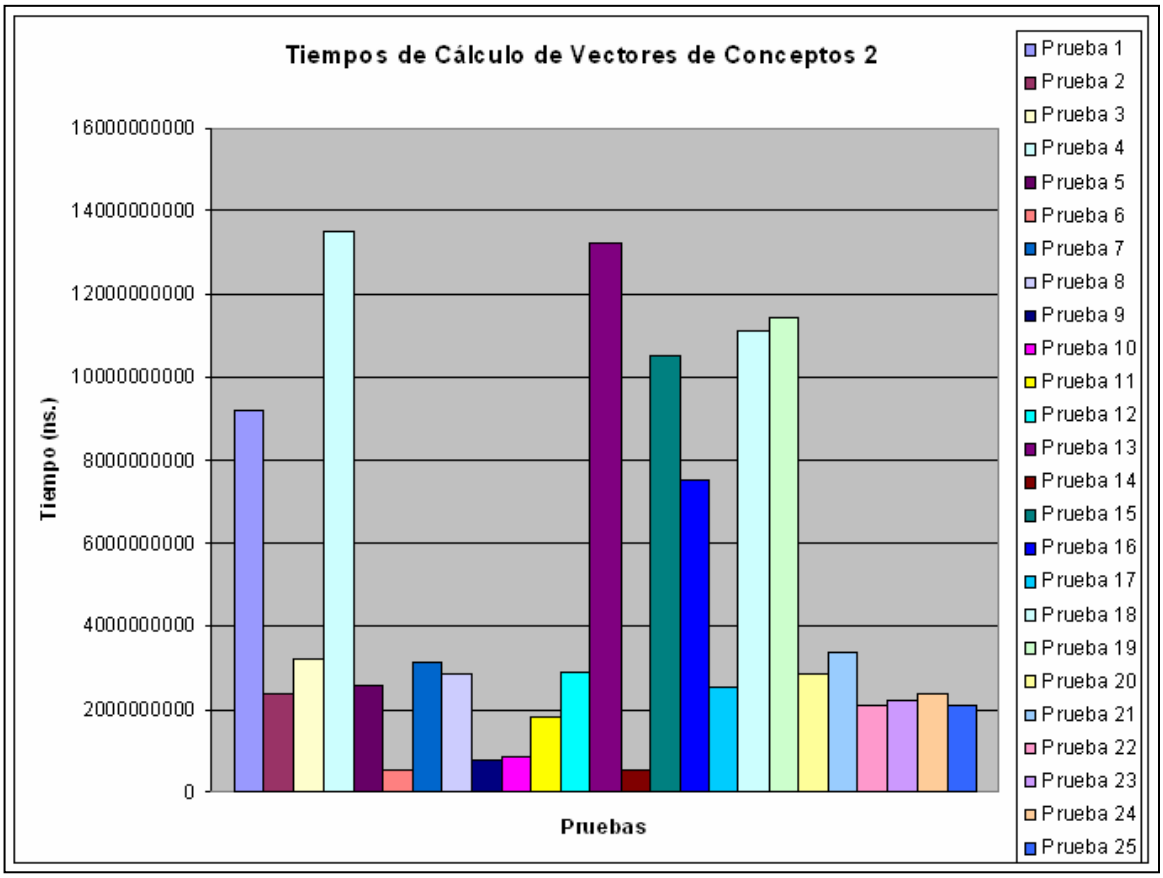


Figura 4.85. Tiempos de Cálculo de Vectores de Conceptos 2 de la Prueba 1 a la 25.

4.2.3.1.1.9. Tiempos de Nombrado de Conceptos.

Este componente se encarga de dar un nombre textual a cada concepto generado durante la ejecución de la creación del modelo de *ACoAR*.

En la figura 4.86 puede verse los tiempos de ejecución del componente de nombrado de conceptos. Cada uno de estos tiempos está directamente relacionado con el número de conceptos generados en su respectiva prueba, durando más tiempo aquellas pruebas con más conceptos generados.

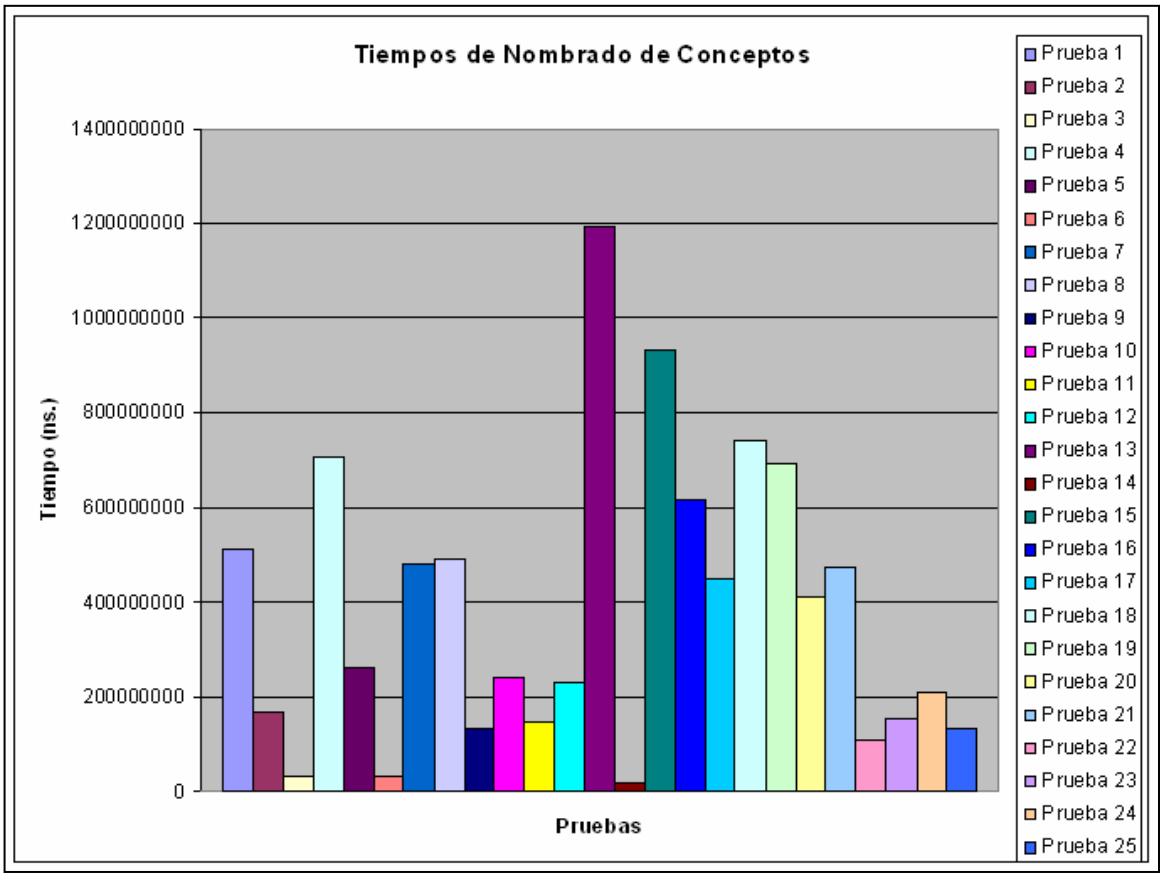


Figura 4.86. Tiempos de Nombrado de Conceptos de la Prueba 1 a la 25.

4.2.3.1.1.10. Tiempo Medio de Cálculo de Similaridad *SCC*.

Este apartado contempla el tiempo medio del cálculo de las similaridades de tipo concepto-concepto de todas las pruebas.

Vemos que casi todas las pruebas tienen un tiempo medio similar, a excepción de las pruebas 3, 6 y 14, que son aquellas que tienen un diccionario con sólo una etiqueta. Aun así, también hay una ligera diferencia entre las pruebas con el parámetro *dictionaryMina* a 100 o a 500 anotaciones, siendo este último grupo el que tiene un tiempo menor que el otro. Así que se podría decir que el tiempo del cálculo de cada similaridad está directamente relacionado con el número de componentes de los vectores.

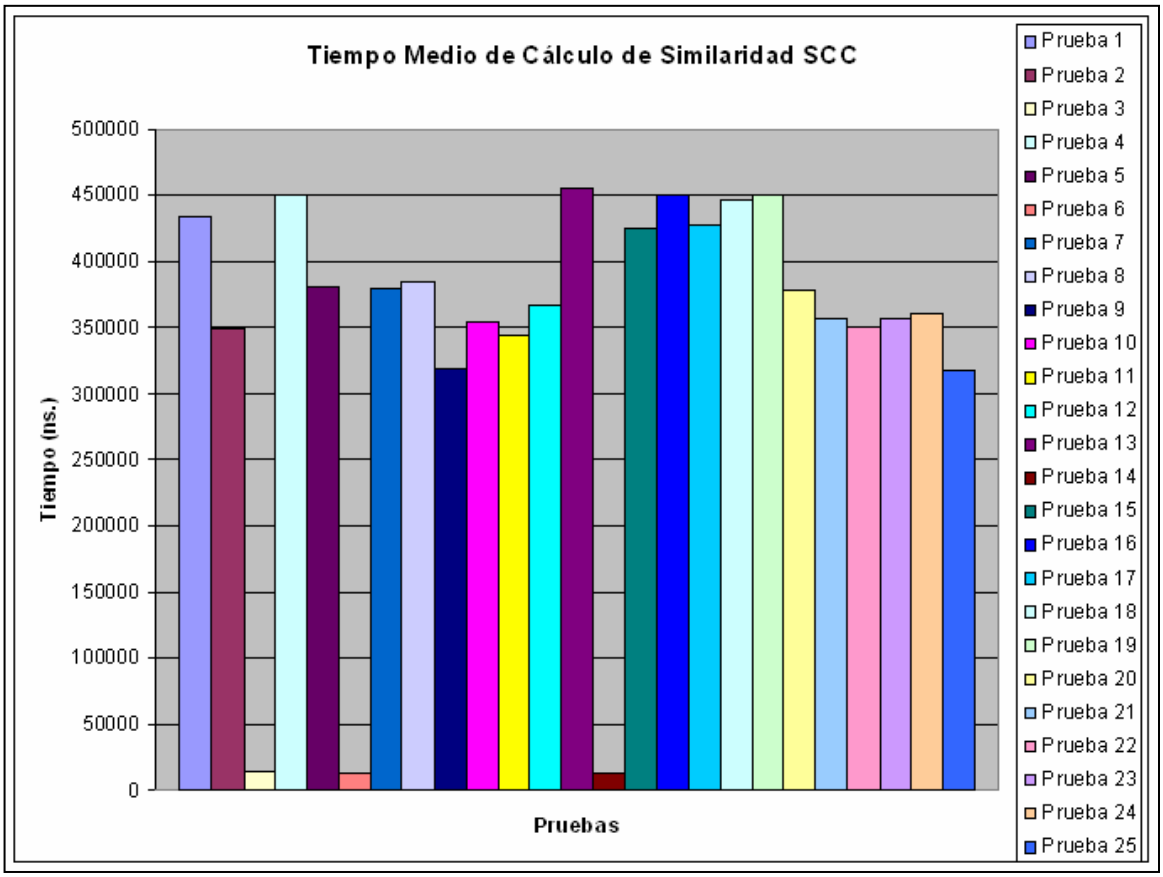


Figura 4.87. Tiempo Medio de Cálculo de Similaridad SCC de la Prueba 1 a la 25.

4.2.3.1.1.11. Tiempo Medio de Inserción *SCC*.

Esta sección muestra el tiempo medio de inserción de las similaridades concepto-concepto en la base de datos.

Hay que tener en cuenta que cada inserción graba en la base de datos todas las similaridades de un concepto con todos los demás. Por tanto no es de extrañar que el tiempo sea mayor en las pruebas que tienen mayor número de conceptos que aquellas que dicho número es inferior.

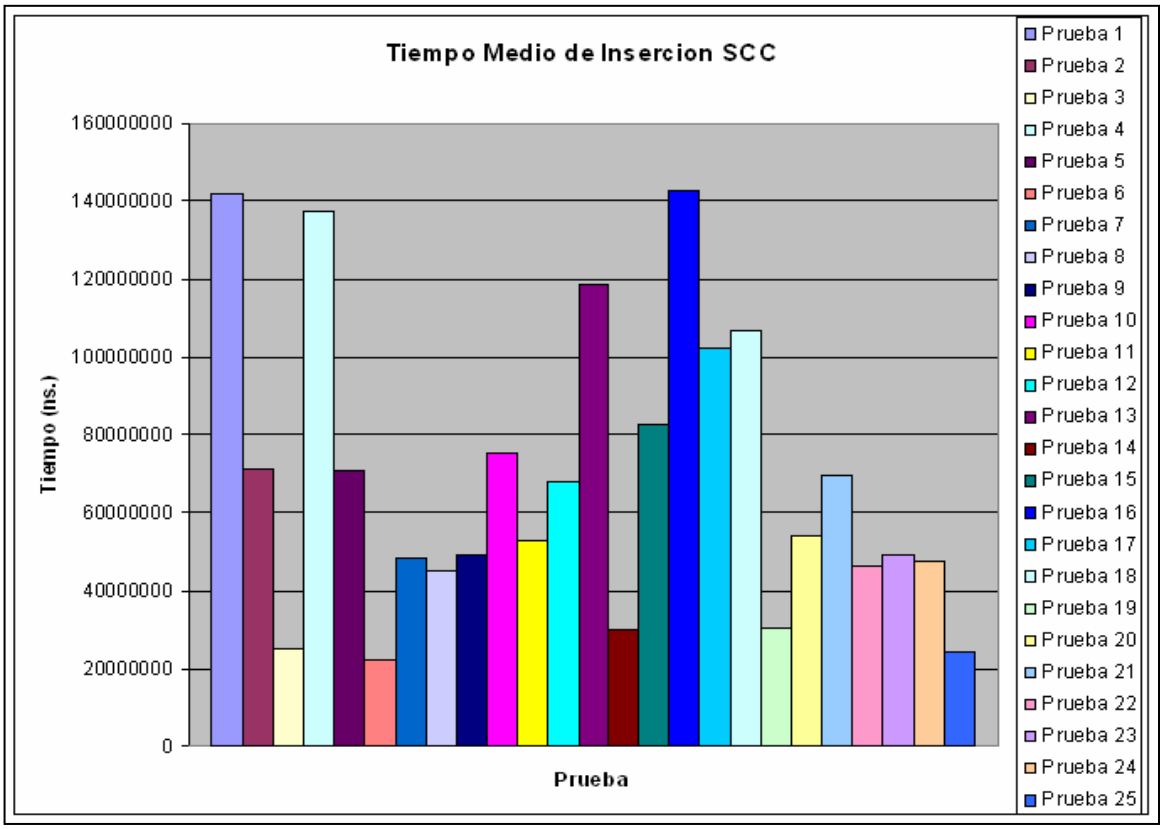


Figura 4.88. Tiempo Medio de Inserción SCC de la Prueba 1 a la 25.

4.2.3.1.1.12. Tiempo Total del Cálculo de *SCC*.

Esta sección trata sobre el tiempo total que le ha costado a cada prueba el cálculo de todas las similaridades concepto-concepto.

En la figura 4.89, puede verse que el tiempo de ejecución está directamente relacionado con el número de conceptos generados en cada prueba.

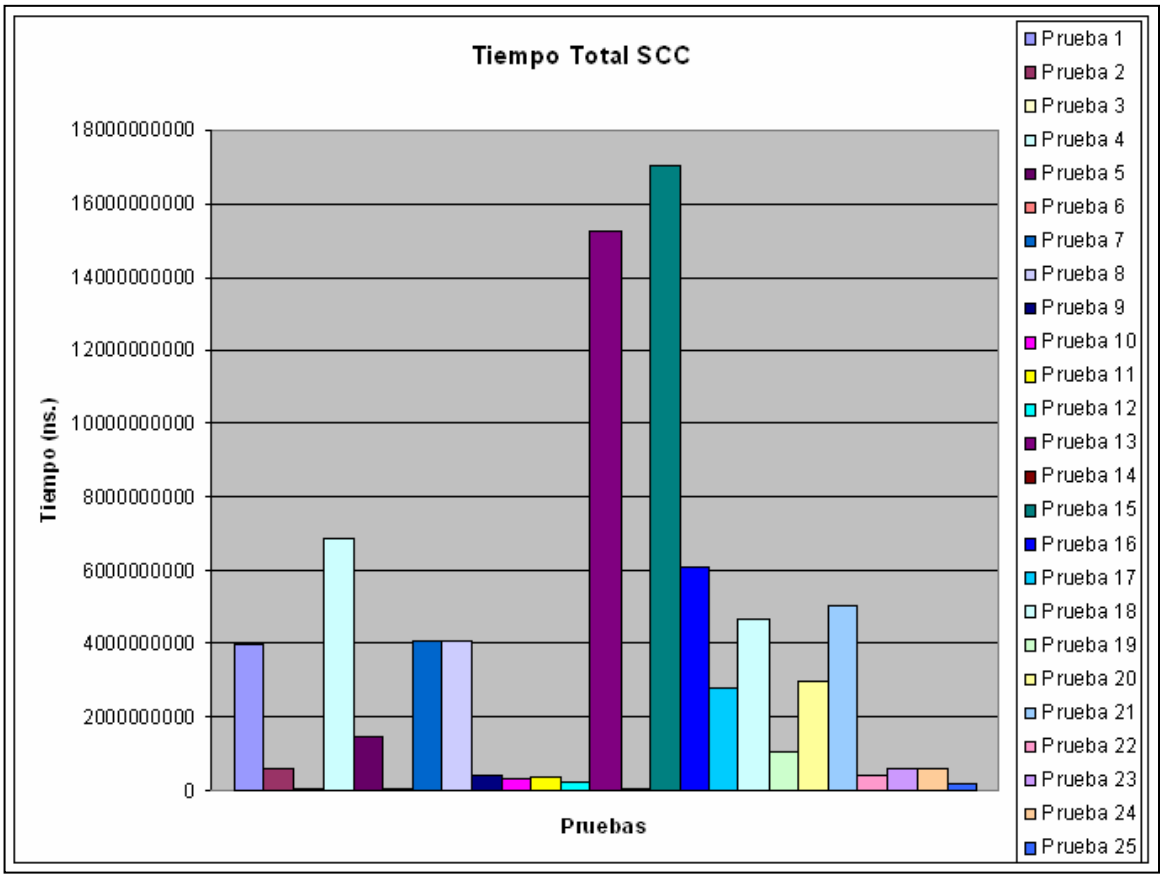


Figura 4.89. Tiempo Total del Cálculo de SCC de la Prueba 1 a la 25.

4.2.3.1.1.13. Tiempo Medio Similaridad *SCD*.

Este apartado se encarga de mostrar el tiempo medio del cálculo de la *similaridad* SCD de cada prueba.

Puede verse, como en la sección 4.2.3.1.1.10, que casi todas las pruebas tienen un tiempo similar a excepción que las pruebas 3, 6 y 14, que muestran un tiempo muy inferior debido a que solamente tienen una etiqueta en el diccionario.

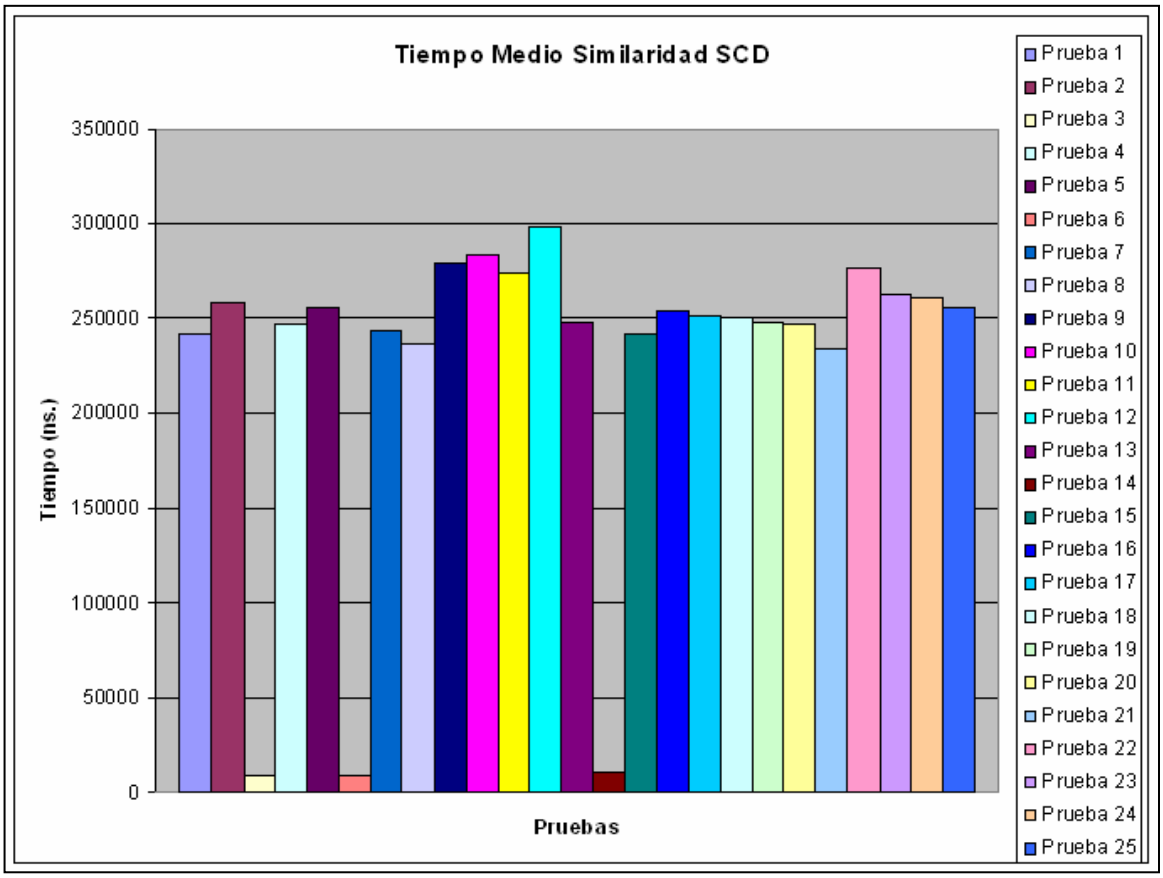


Figura 4.90. Tiempo Medio del Cálculo de SCD de la Prueba 1 a la 25.

4.2.3.1.1.14. Tiempo Medio de Inserción de *SCD*.

Esta sección enseña el tiempo medio empleado en cada prueba para la inserción de las similaridades SCD en la base de datos.

En este caso los tiempos medios están relacionados con el número de etiquetas que hay en el diccionario, ya que cada inserción en la base de datos guarda todas las similaridades de un concepto con todas las etiquetas del diccionario.

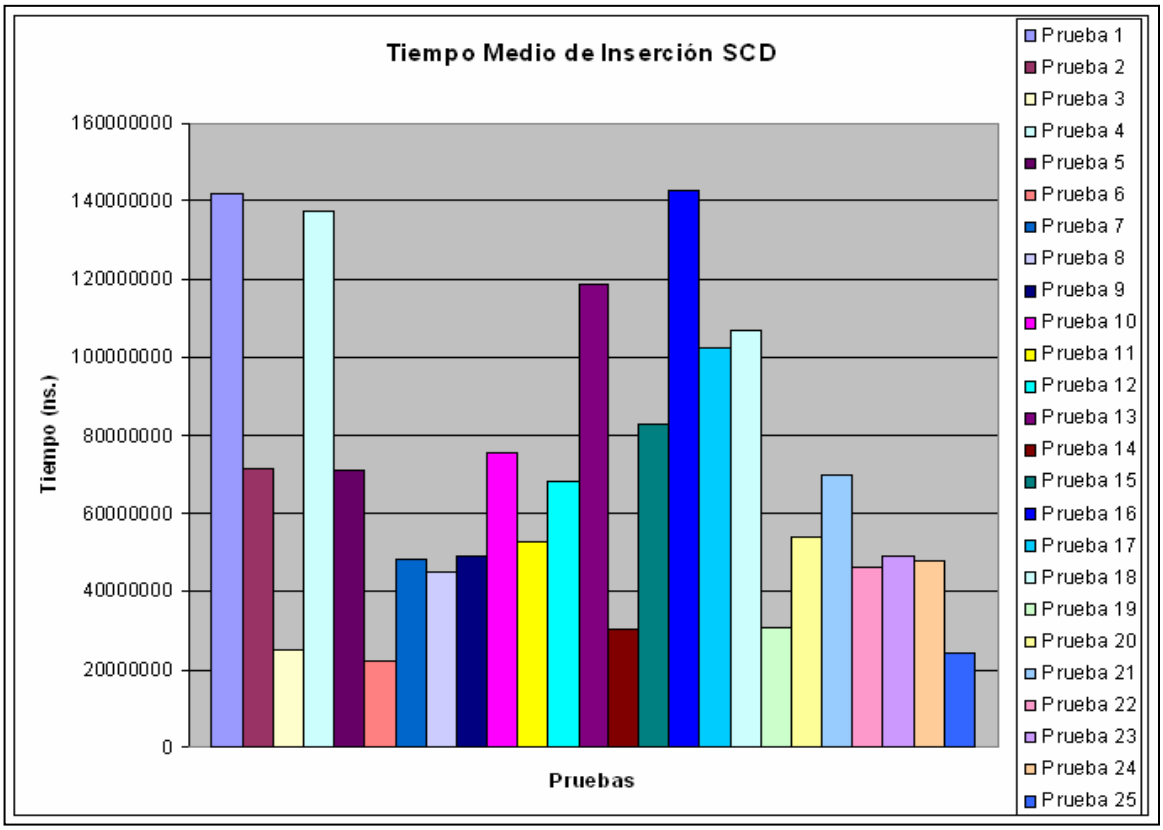


Figura 4.91. Tiempo Medio de Inserción de SCD de la Prueba 1 a la 25.

4.2.3.1.1.15. Tiempo Total del Cálculo de *SCD*.

Esta sección se encarga de enseñar el tiempo total que emplea cada prueba para calcular todas las similaridades de tipo concepto-diccionario.

A consecuencia de las conclusiones llegadas en los dos apartados anteriores, podemos concluir que el tiempo total del cálculo de SCD está directamente relacionado con el número de etiquetas del diccionario, pero también está directamente relacionado con el número de conceptos generados en cada prueba.

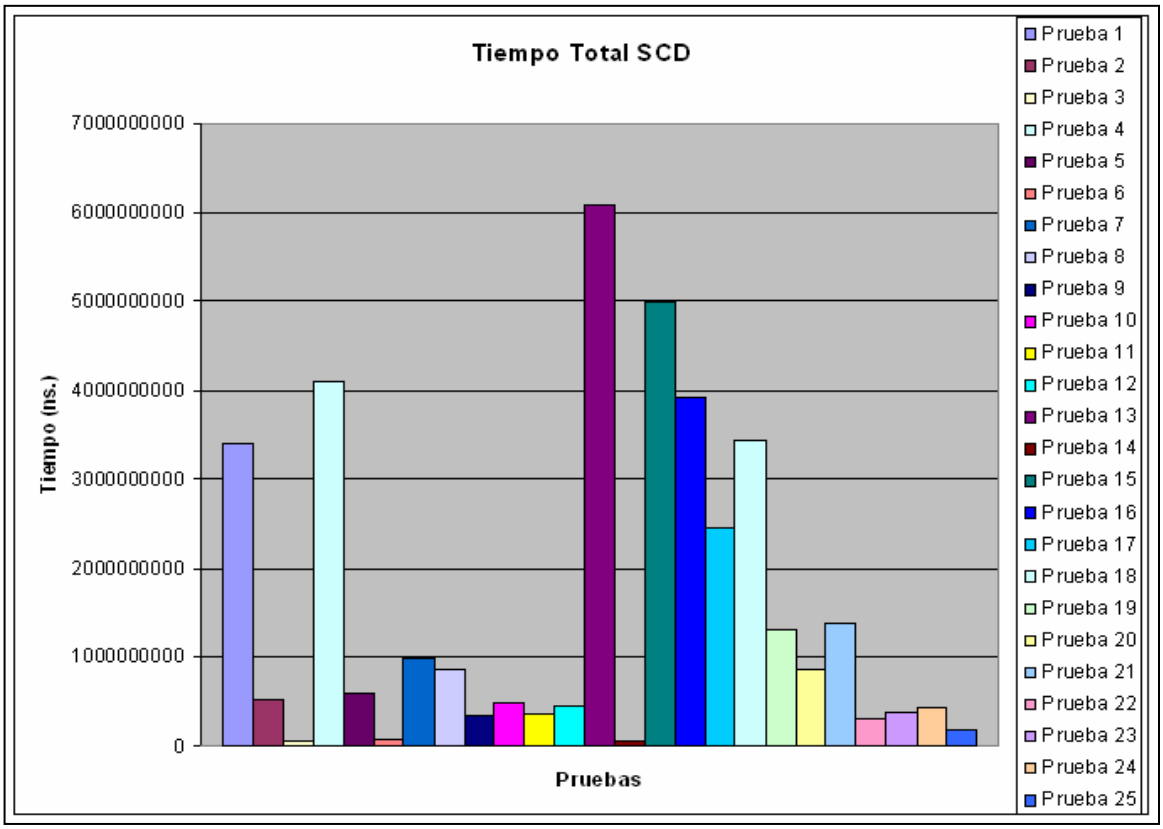


Figura 4.92. Tiempo Total del Cálculo de SCD de la Prueba 1 a la 25.

4.2.3.1.1.16. Tiempo Medio del Cálculo de *SCR*.

Como en otros apartados tratando tiempos medios de otras similaridades, puede observarse en la figura 4.93, que casi todas las pruebas tienen un tiempo medio similar. En este caso las excepciones las producen las pruebas 13, 15.

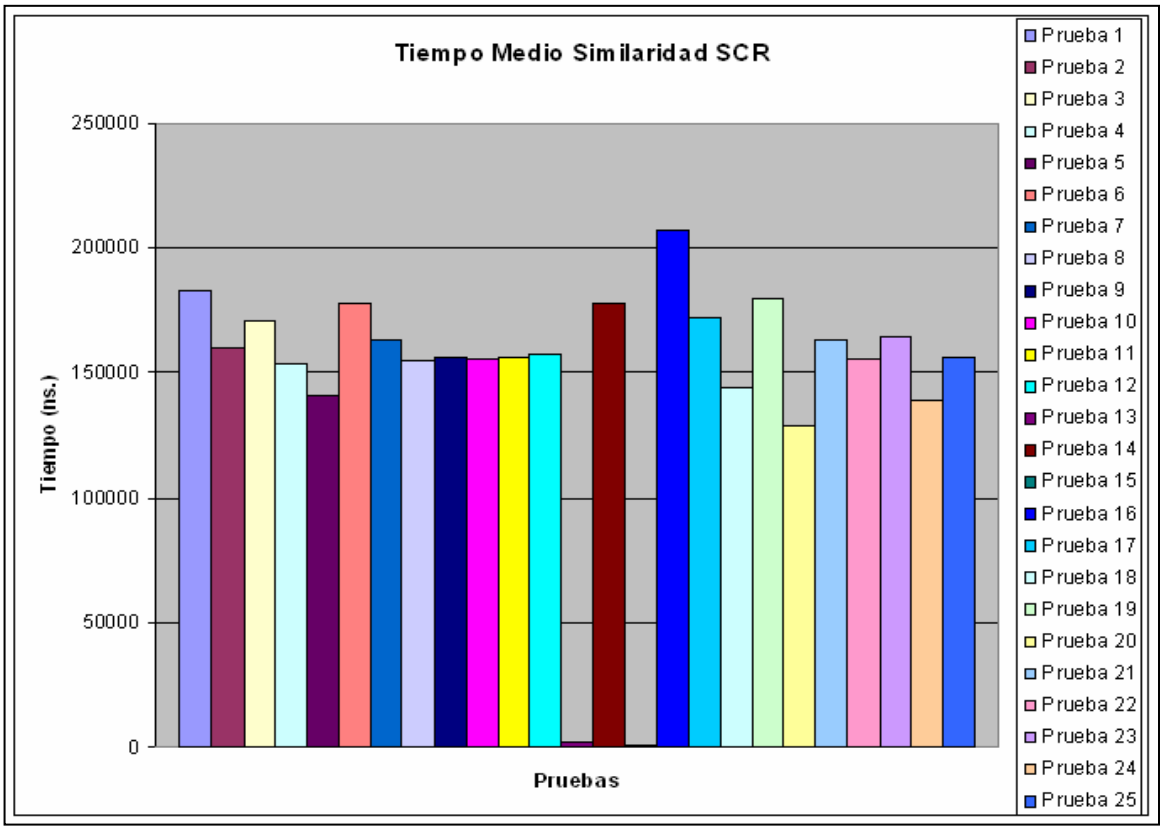


Figura 4.93. Tiempo Medio del Cálculo de SCR de la Prueba 1 a la 25.

4.2.3.1.1.17. Tiempo Medio de Inserción de *SCR*.

En esta sección se trata con el tiempo medio de cada prueba para la inserción de las similaridades concepto-recurso en la base de datos.

Cada inserción en la base de datos almacena las similaridades entre un concepto y todos los recursos. Según esto, el tiempo de inserción debiera ser más o menos homogéneo, pero no sucede así. Esto puede deberse a que en algunas de las pruebas surjan muchas similaridades igual a cero, las cuales no son insertadas, reduciendo con ello el vector de inserción en la base de datos y con ello el tiempo de inserción del mismo.

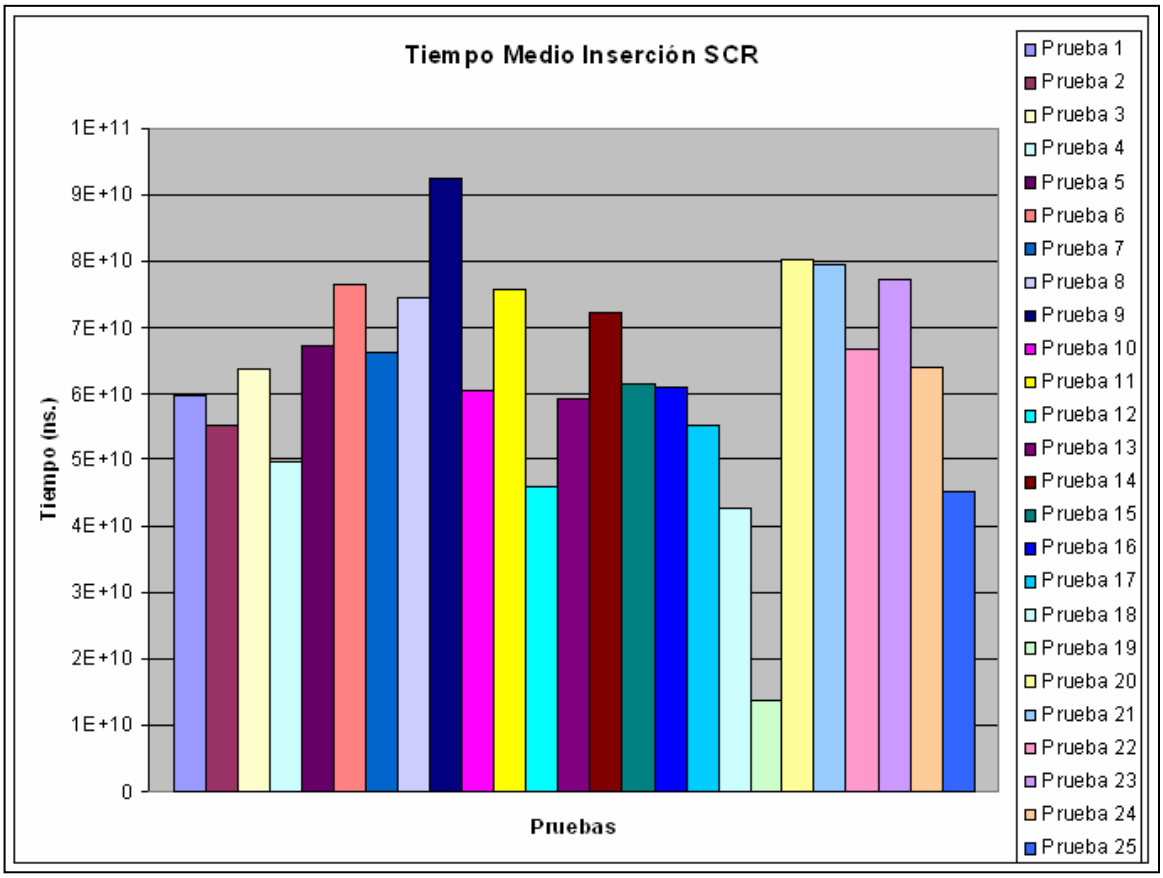


Figura 4.94. Tiempo Medio de Inserción de SCR de la Prueba 1 a la 25.

4.2.3.1.1.18. Tiempo Total del Cálculo de *SCR*.

Esta sección recoge el tiempo total que se ha empleado en cada prueba para el cálculo de las similaridades concepto-recurso.

En la gráfica 4.95 puede observarse que la mayor parte de las pruebas con un tiempo alto son aquellas con un número de conceptos generados elevado. Por tanto, el número de conceptos es directamente proporcional al tiempo total de cálculo de las similaridades concepto-recurso.

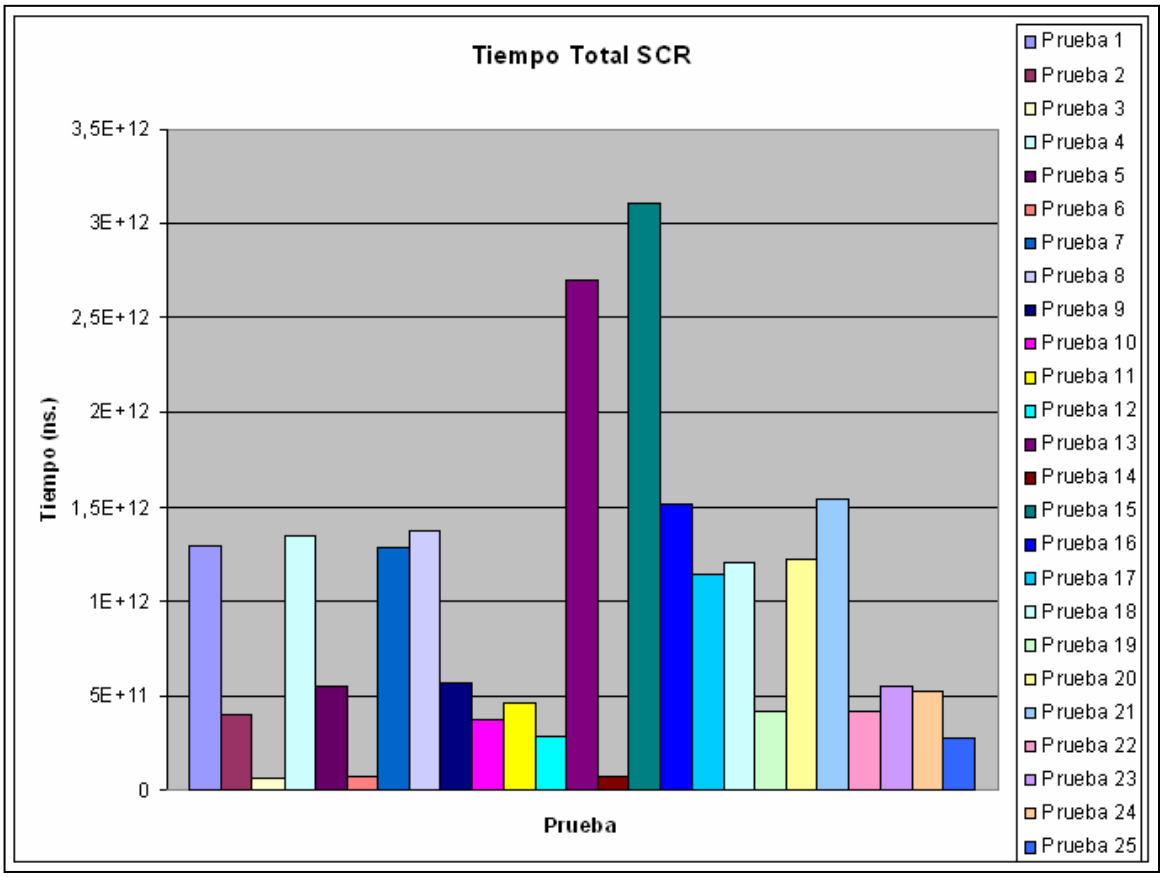


Figura 4.95. Tiempo Total del Cálculo de SCR de la Prueba 1 a la 25.

4.2.3.1.1.19. Tiempo Medio del Cálculo de *SRR*.

Esta sección detalla el tiempo medio del cálculo de las similaridades recurso-recurso de cada.

Puede verse en la figura 4.96 que las pruebas con mayor tiempo medio son aquellas con el parámetro dictionaryMina configurado a 100 anotaciones. Por tanto, debe haber una relación directa entre el tamaño del diccionario y el tiempo en el cálculo de una SRR.

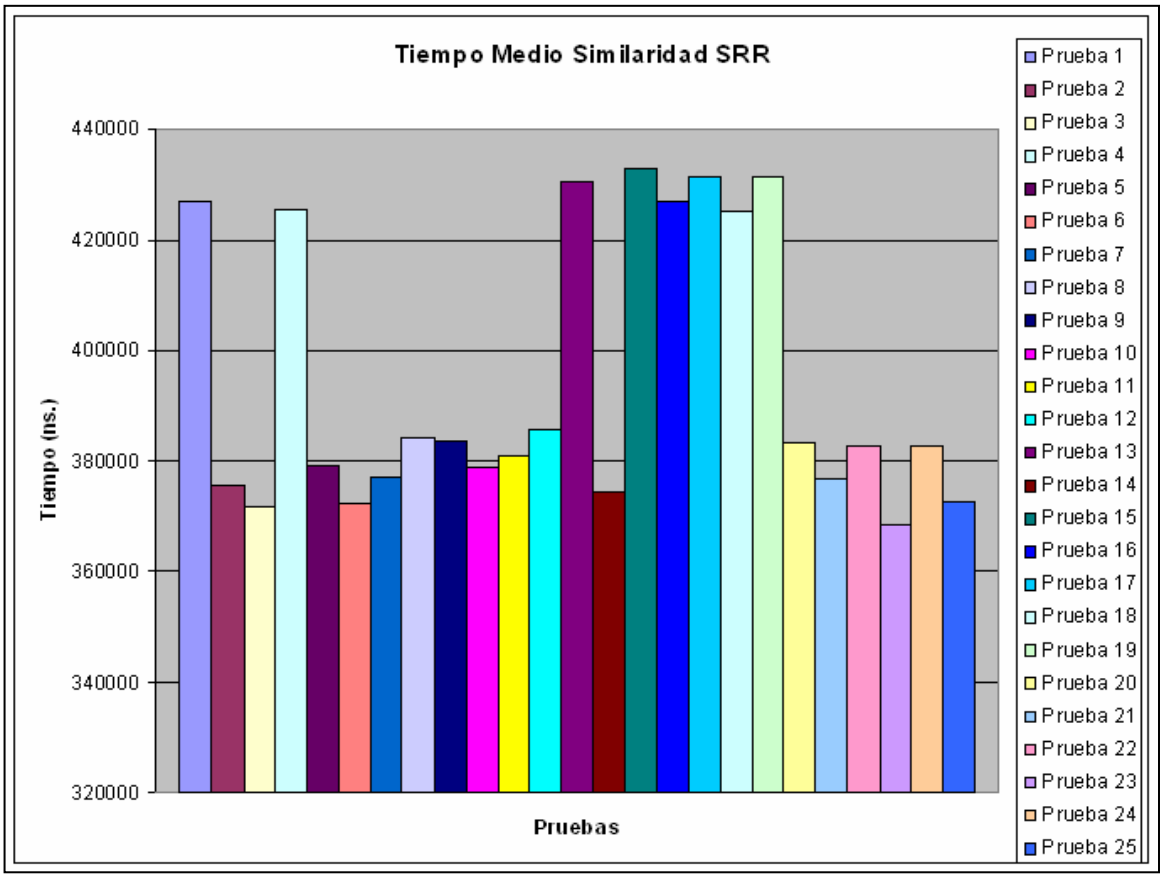


Figura 4.96. Tiempo Medio del Cálculo de Similaridad SRR de la Prueba 1 a la 25.

4.2.3.1.1.20. Tiempo Medio la Inserción de *SRR*.

Aquí se detalla los tiempos medios de inserción en la base de datos de las similaridades recurso-recurso de todas las pruebas.

Según la figura 4.97, parece que hay diferencias notables entre las distintas pruebas, pero estas en realidad son ínfimas, comparadas con otras inserciones de similaridades, ya que en esta la media representa la inserción de una similaridad individual en la base de datos.

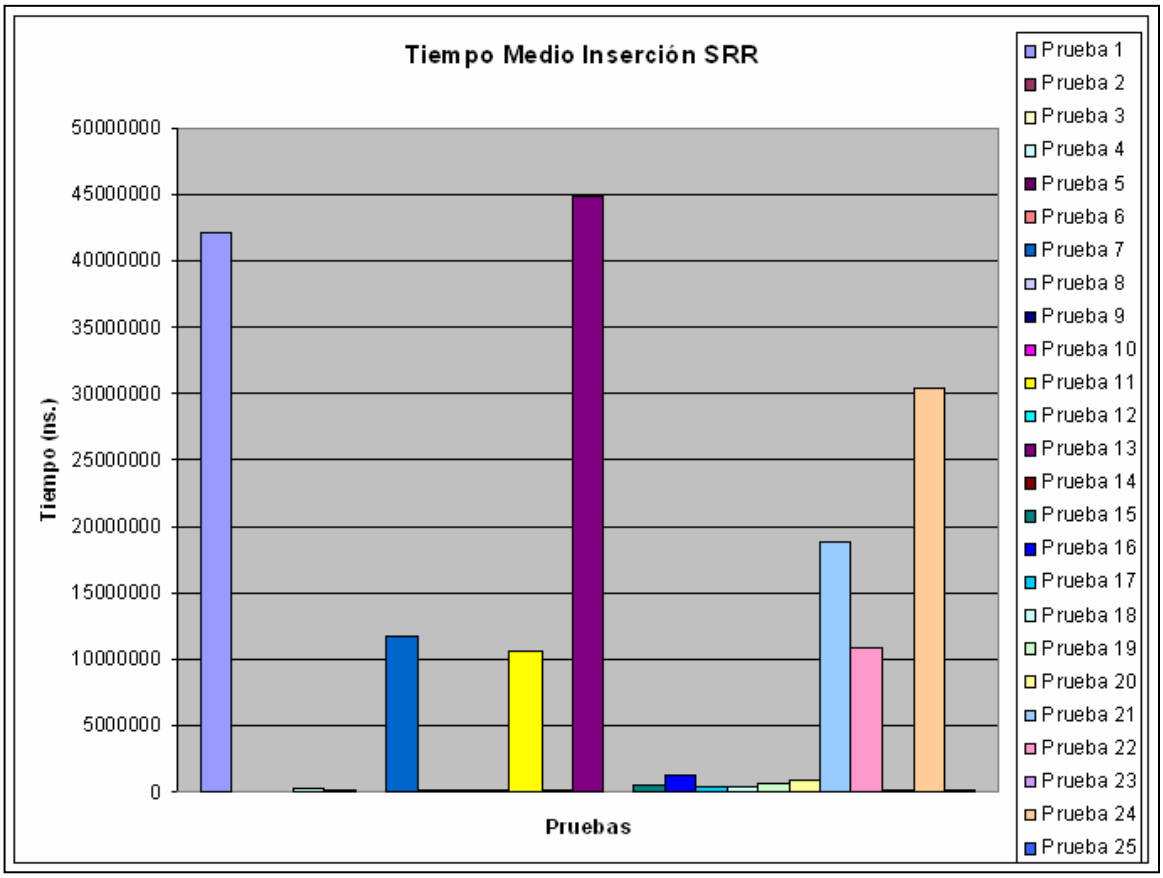


Figura 4.97. Tiempo Medio de Inserción de SRR de la Prueba 1 a la 25.

4.2.3.1.1.21. Tiempo Medio la Inserción de *SRR*.

Esta sección muestra los tiempos totales del cálculo de todas las similaridades recurso-recurso de cada prueba.

Estos tiempos se ven altamente influenciados por el número de conceptos de cada prueba, y los que son suficientemente similares, definidos por el parámetro *similaritiesSRRThres*, ya que solo se comparan entre sí los recursos pertenecientes a un mismo concepto, o a conceptos que sean suficientemente similares.

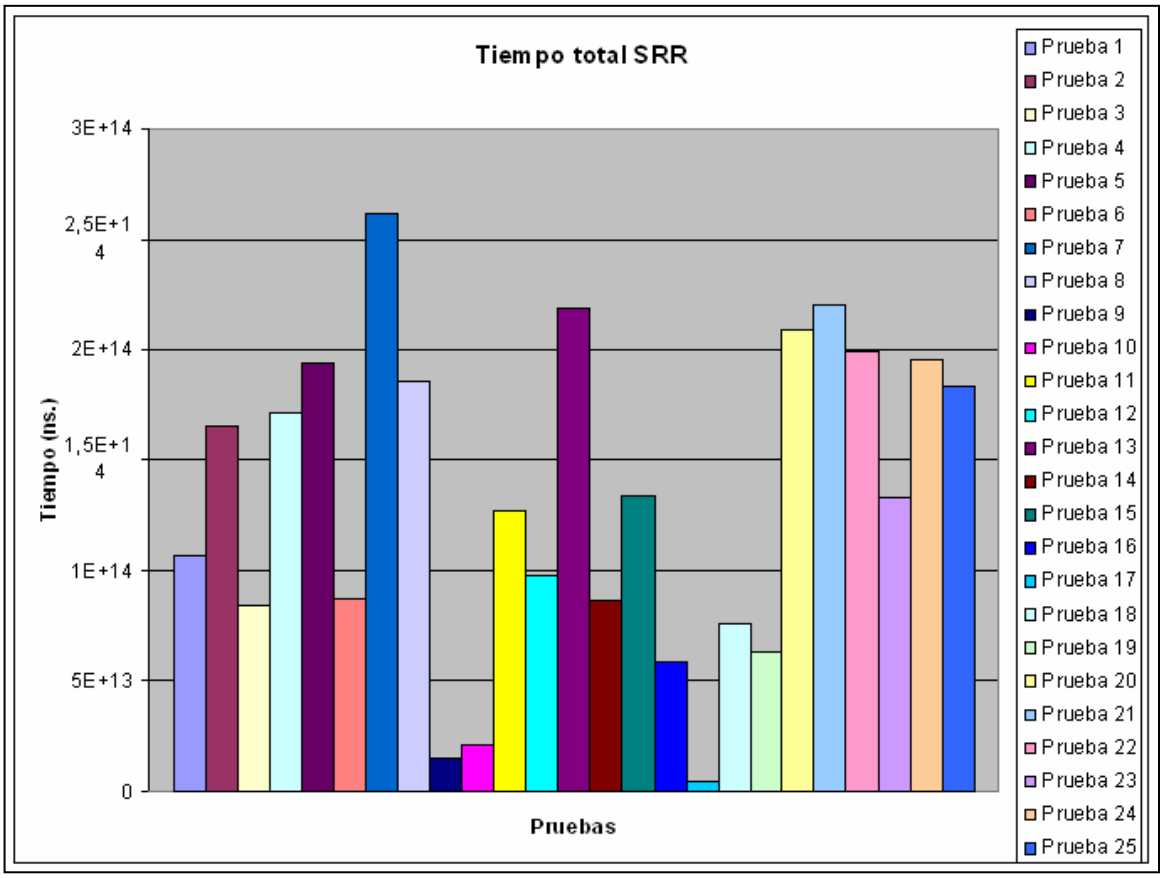


Figura 4.98. Tiempo Total del Cálculo de SRR de la Prueba 1 a la 25.

4.2.3.1.1.22. Tiempo Total de Ejecución.

Aquí se detalla el tiempo de ejecución total de cada prueba.

Observando la figura 4.98 y la anterior, referente al cálculo de las similaridades SRR, puede llegarse a la conclusión de que son prácticamente idénticas. Esto lleva a afirmar que prácticamente el tiempo total de ejecución de la aplicación es ocupado por el cálculo de las similaridades recurso-recurso en todas las pruebas realizadas.

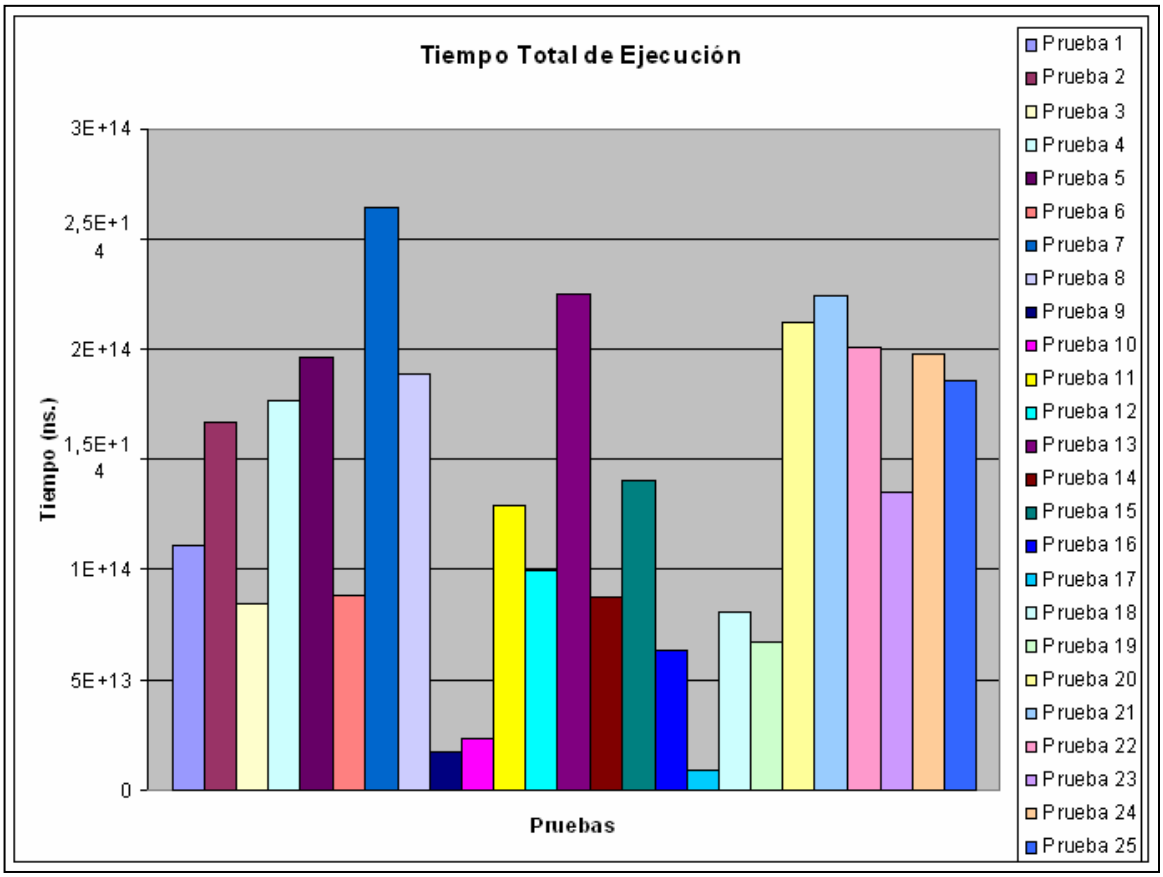


Figura 4.99. Tiempo Total de Ejecución de la Prueba 1 a la 25.

4.2.3.1.2. Estadísticas de Clasificación de Recursos.

Esta sección se centra en las estadísticas de los conceptos, tanto su número en cada prueba, como los recursos clasificados en ellos y los no clasificados.

4.2.3.1.2.1. Número de Conceptos en cada Prueba.

En la figura 4.100 puede verse que ha habido una gran variedad de número de conceptos generados en las diferentes pruebas. Por lo general, aquellas con un valor menor de *dictionaryMina* han generado un número mayor de conceptos y viceversa. También se nota un aumento en el número de conceptos al elevar el valor de *mergingThreshold*.

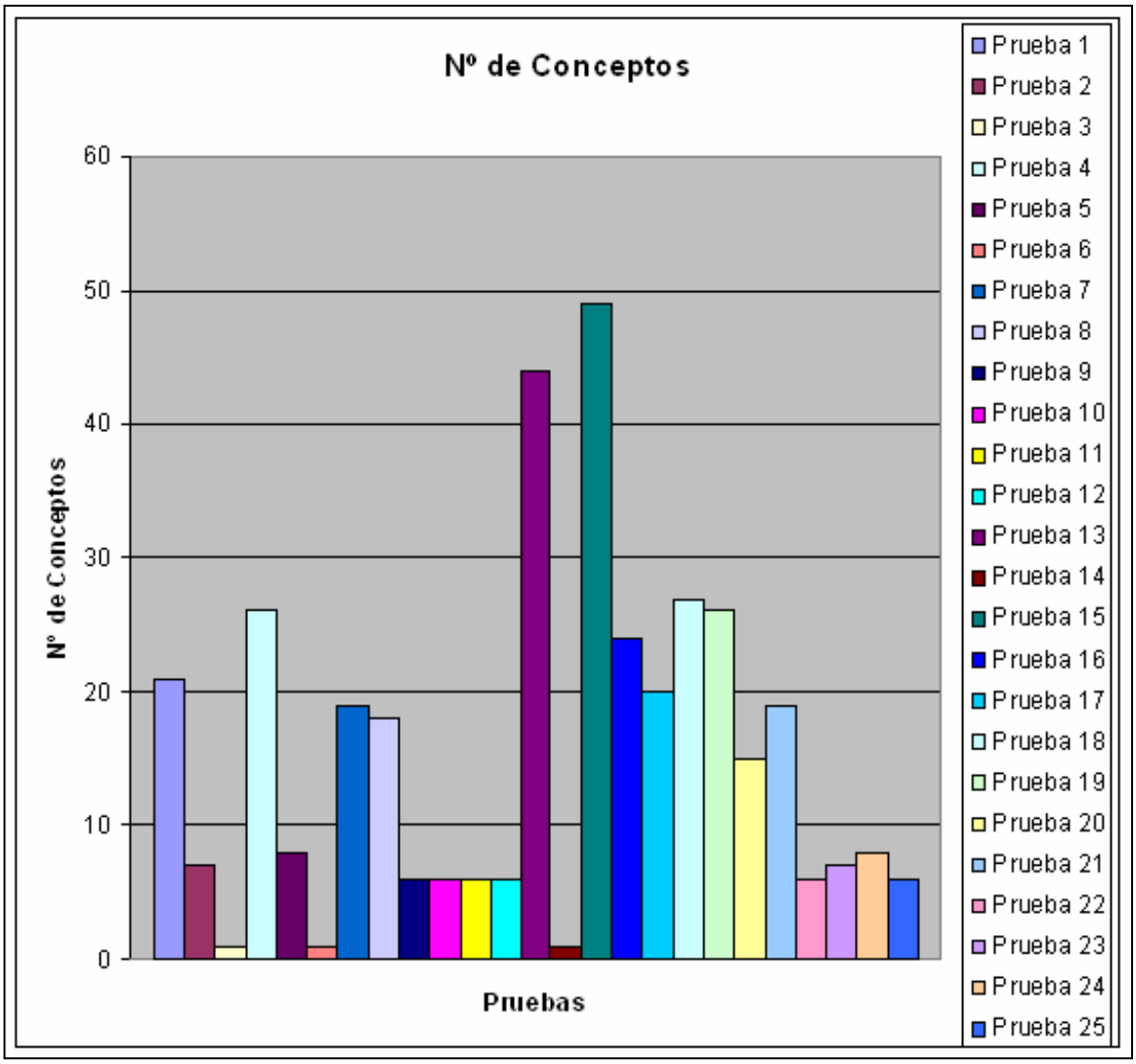


Figura 4.100. Número de Conceptos de la Prueba 1 a la 25.

4.2.3.1.2.2. Clasificación de Recursos.

En la figura 4.101 puede verse que en la mayor parte de las pruebas ha habido una mayoría de recursos clasificados frente a los que se han quedado sin pertenecer a algún concepto. De entre las pruebas que han tenido un mayor número de recursos no clasificados están aquellas con el parámetro *dictionaryMina* configurado a 2564 (pruebas 3, 6 y 14), algo que era de esperar ya que solamente podían se clasificados los recursos con la etiqueta *ultrasound*. Las otras pruebas con alto número de recursos no clasificados (pruebas 9, 10 y 17) coinciden en que el parámetro *classifierThreshold* toma valores por encima de 0.1, el valor estándar definido para las pruebas. También habría que destacar que parece ser que hay una mayor cantidad de recursos clasificados en pruebas con el clasificador *Sim* que con el clasificador *Delta*.

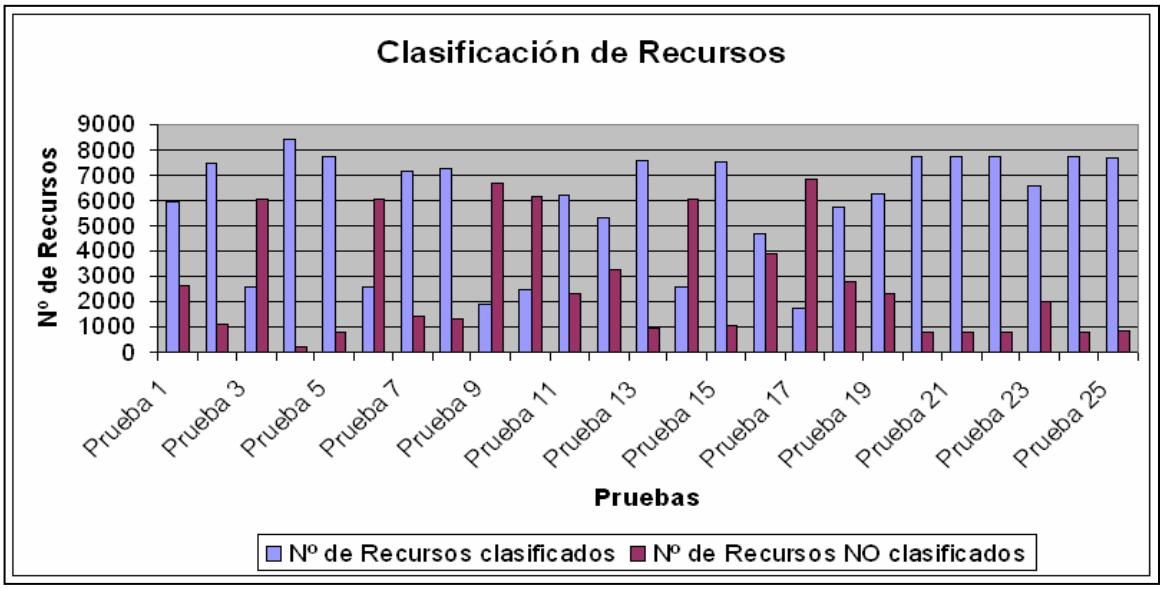


Figura 4.101. Clasificación de Recursos de la Prueba 1 a la 25.

4.2.3.1.2.3. Recursos *Converged* y *Pending*.

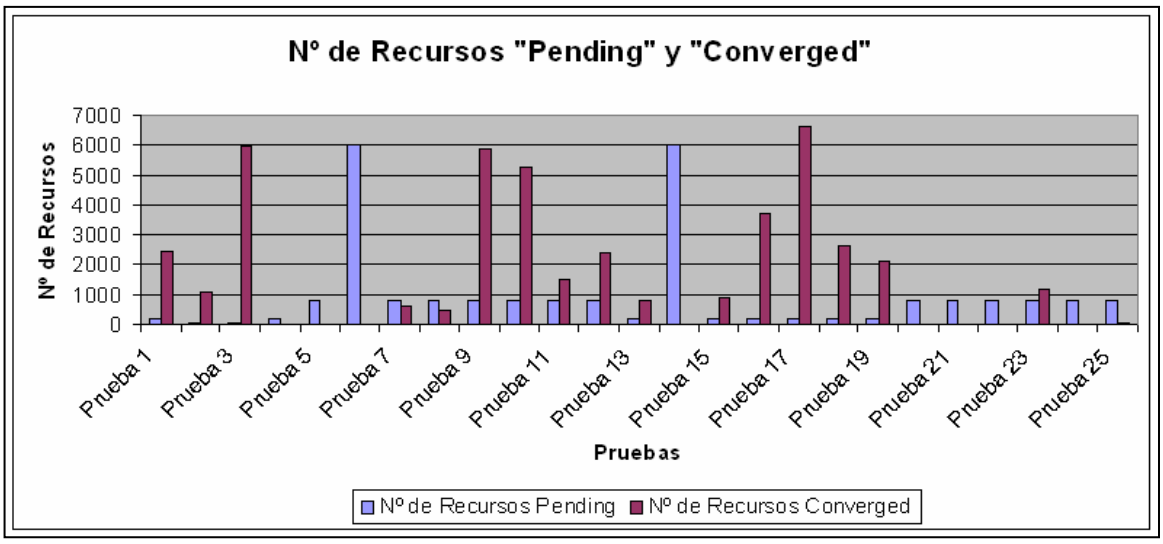


Figura 4.102. Número de Recursos Pending y Converged.

Los recursos no clasificados se dividen en dos clases, los recursos *coverged*, que son aquellos que han convergido pero que no han podido ser clasificados, y los recursos *pending*, aquellos que ni siquiera han podido llegar a converger.

Cabe destacar que aquellas pruebas con *dictionaryMina* a 2564 anotaciones, la totalidad de los recursos no clasificados son *pending*.

4.2.3.2. Validación de la Clasificación.

Para comprobar que los recursos están bien clasificados en su correspondiente concepto, se ha hecho uso de dos *testers* para que lo comprueben manualmente. Cada uno de ellos ha *testeado* los resultados de un subconjunto de las pruebas totales. Este análisis se ha realizado obteniendo el 4% de las imágenes médicas clasificadas de cada prueba y mostrándoselas a los *testers* junto con el concepto al que pertenece cada una de ellas. Cada una de las imágenes mostradas ha sido valorada entre 1 (lo peor) y 5 (lo mejor) respondiendo a lo mal o bien que está dicha imagen clasificada en el concepto según la opinión de la persona que la estuviese valorando.

A continuación se muestran una serie de gráficas con los datos obtenidos de la forma anteriormente indicada.

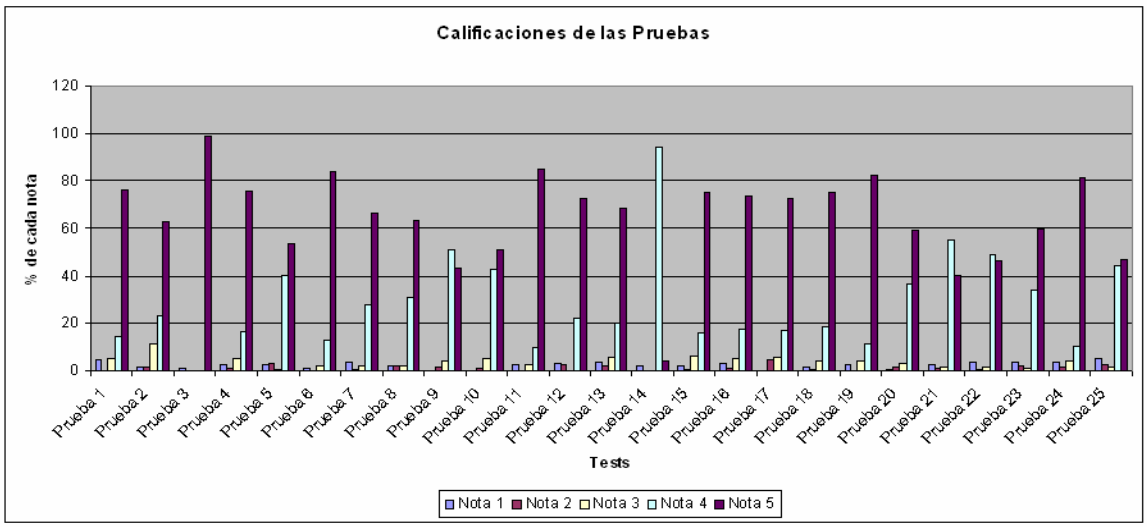


Figura 4.103. Calificación de las Pruebas.

La figura 4.103 muestra el porcentaje de recursos con su clasificación valorada con cierta nota de las muestras tomadas para realizar los *tests*. Como puede verse todas las clasificaciones han sido valoradas en su gran mayoría con 4 o 5. Según esto puede considerarse que los recursos están bien clasificados en sus respectivos conceptos según las personas realizaron las pruebas de validación.

Por otro lado, la figura 4.104, indica la nota media de cada una de las pruebas de validación. Puede observarse que para la mayoría las pruebas la nota media está entre 4 y 5, por tanto puede decirse que la clasificación de los recursos ha sido satisfactoria en todas las pruebas según la opinión de los *testers*.

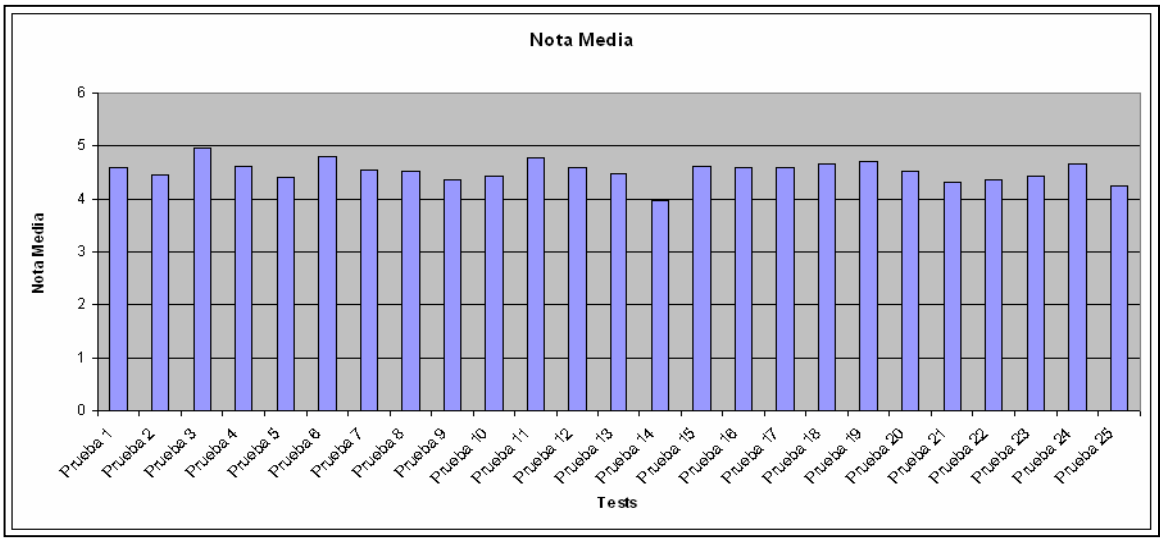


Figura 4.104. Nota Media de los Test de Validación.

4.2.4. Conclusiones y comparación con otras propuestas.

Como ha podido apreciarse durante el estudio de las diferentes pruebas, las distintas configuraciones de los parámetros de esta versión de *ACoAR* influyen en los tiempos de ejecución y en los resultados obtenidos.

El parámetro *dictionaryMina*, responsable de definir el número de anotaciones que debe tener una etiqueta para pertenecer al diccionario, tiene bastante influencia en el desarrollo de la ejecución de las pruebas. Cuanto más bajo sea el valor de este parámetro, mayor va a ser el diccionario y, por tanto, mayor va ser el espacio vectorial donde se van a representar los recursos, conceptos y etiquetas. Esto va a afectar directamente en el tiempo de ejecución de ciertos componentes del algoritmo. Cuanto mayor es el diccionario mayor es el tiempo de creación del mismo, de *clustering*, de cálculo de vectores y de cálculo de similaridades.

También el tamaño del diccionario influye en el número de conceptos generados en cada prueba. Cuanto mayor es el número de etiquetas que están incluidas en el diccionario, mayores oportunidades hay de que se creen conceptos más específicos para clasificar los mismos recursos, y, por tanto, se crean más conceptos. El número de conceptos generados también se ve influenciado por el parámetro *mergingThreshold*, que es el encargado de establecer la similaridad mínima para que dos conceptos se consideren similares y sean transformados en uno solo. En las pruebas puede apreciarse como aumenta el número de conceptos cuanto mayor es el valor asignado a dicho parámetro.

Cuando el tamaño del diccionario es muy pequeño la gran mayoría de los recursos se quedan sin clasificar. Esto se debe que no hay suficientes etiquetas de diccionario para crear un número de conceptos mínimo con el que representar siquiera los grupos más genéricos de recursos del sistema. El parámetro *classifierThreshold*, que indica la similaridad mínima entre un recurso y su concepto para considerar que está bien clasificado, también influye en la cantidad de recursos clasificados. Puede observarse en las estadísticas que al asignarle un valor por encima de 0.1 se obtiene un porcentaje mayor

de recursos no clasificados que aquellos clasificados, al menos en pruebas que se ha usado el clasificador *Delta*. Esto parece indicar que, en aquellas pruebas con clasificador *Delta*, los recursos son más fácilmente considerados como mal clasificados.

Las pruebas con clasificador *Sim* tienen un porcentaje mayor de recursos clasificados. Esto es debido a que el clasificador *Delta* es más estricto que el clasificador *Sim* y, por tanto, es más fácil que considere un recurso como mal clasificado en su concepto.

Uno de los fenómenos que se producen en pruebas con un diccionario grande es que se crean ciertos conceptos con muchas etiquetas y algunas, a priori, no parecen tener mucho que ver entre sí. Esto también se ve reflejado en el nombre que toma el concepto. Un ejemplo de ello es el concepto "*mri & brain & knee & magnetic & resonance & acl & 2009 & torn & houston & texas & left & or & of & magneticresonanceimaging & bones & radiology & broken & bone*" generado en la prueba 1. Como puede observarse, en él hay etiquetas que no parecen tener mucha relación, como *brain* y *knee*, y otras que pueden ser consideradas como ruido como *or* y *of*. Estos fenómenos también se producen en pruebas con diccionarios más pequeños, pero apareciendo de forma más errática.

Hay otros factores diferentes a los parámetros de configuración, aunque relacionados con ellos, que influyen también en los resultados y en los tiempos de ejecución. El número de recursos está directamente relacionado con los tiempos de ejecución empleados para comprobar la correcta clasificación de los mismos y el tiempo total para calcular todas las similaridades concepto-recurso y recurso-recurso. Este último tipo de similaridades también está influenciado por el número de conceptos y el grado de similaridad entre ellos, ya que sólo se calcula esta similaridad entre los recursos de un mismo concepto y de aquellos conceptos que se consideren suficientemente similares.

El número de conceptos generados durante la ejecución afecta a los tiempos de ejecución de nombrado de conceptos y del cálculo total de similaridades concepto-concepto, siendo mayores cuantos más conceptos hay en el sistema.

El número de etiquetas del diccionario está directamente relacionado con la cantidad de tiempo empleado para el cálculo de similaridades concepto-diccionario.

Del tiempo total de ejecución de todas las pruebas llevadas a cabo en este proyecto, la mayor parte ha sido empleado para el cálculo de las similaridades recurso-recurso. Esto no es a causa de la complejidad de las operaciones en sí, sino del gran número de combinaciones de recursos que hay que realizar.

Según las pruebas de validación hechas por los *testers*, en todas las pruebas, los recursos están correctamente clasificados en sus conceptos. Sobre este resultado hay que tener en cuenta que las personas que realizaron las pruebas de validación, ninguna estaba relacionada con la medicina, así que puede haber habido ciertos fallos o puede que no hayan sido valoradas como lo haría el usuario final.

Como conclusión quisiera señalar ciertos aspectos. Se ha conseguido crear, en las diferentes pruebas, modelos de *ACoAR* con una clasificación de recursos bastante satisfactoria. Cada uno de estos modelos tiene diferente número de conceptos, pero la mayoría con un alto porcentaje de clasificación de recursos. Ahora es cuestión de decidir que tipo de resultado es el que se ajusta más al uso final que va a darse al modelo generado. Igual para un fin es más acertado tener los recursos clasificados en pocos

conceptos y más genéricos y, para otro, tener una gran cantidad de conceptos más específicos; igual es necesario un modelo con una clasificación más estricta u otro con una más flexible, etc.

Uno de los problemas que veo a la hora de trabajar con conceptos muy específicos, en base a la folksonomía de base que he empleado, es la inclusión de ruido en la definición de los conceptos, con esto me refiero a que aparecen etiquetas que o no tienen contenido semántico, como "*or*" y "*of*", o que igual no tienen mucho que ver con el tema tratado, el de las imágenes médicas. Por eso, para el dominio aquí tratado sobre medicina, creo que sería conveniente quitar aquellas etiquetas que no interesan o que distorsionan los cálculos y los resultados. Lo ideal sería hacerlo de forma automática, por ejemplo usando herramientas como *Wordnet*, para quedarnos sólo con etiquetas con contenido semántico, o *Snomed-CT*, para quedarnos sólo con términos médicos.

Otro problema que veo es el tiempo requerido para el cálculo de las similaridades recurso-recurso, que es el que hace, al menos en las pruebas aquí realizadas, que el tiempo de ejecución de la aplicación se tan alto. Por tanto, yo aconsejaría realizar estos cálculos sólo cuando sea estrictamente necesario para los fines que se va a emplear el modelo resultante, o, si no, buscar otra forma de hacer los cálculos de una manera más rápida, como por ejemplo hacer uso de la programación distribuida.

En resumen, se han generado una serie de modelos que han clasificado los recursos de forma satisfactoria, según la opinión de las personas que los han validado, pero que presenta ciertos problemas a la hora de su creación, como el tiempo de ejecución, o el uso de etiquetas no semánticas o no relacionadas con imágenes médicas.

5. Conclusiones y Líneas Futuras.

5.1. Conclusiones.

Tras finalizar la ejecución de este proyecto se puede concluir que se han alcanzado todos los objetivos planteados en un principio, y algunos otros que han ido surgiendo a lo largo del mismo.

Se ha conseguido construir una aplicación que es capaz de clasificar los recursos de una folksonomía de imágenes médicas en una serie de conceptos haciendo uso del trabajo desarrollado en la tesis doctoral de Francisco Echarte [4]. La aplicación ha sido desarrollada en *Java*, un lenguaje orientado a objetos, utilizando las características del mismo para construir cada componente del algoritmo como un módulo independiente. Además al haber sido desarrollado en este lenguaje le dota la característica de ser multiplataforma, lo que facilita su portabilidad junto con el hecho de que al compilarlo todo queda desplegado en una única carpeta (a excepción de la base de datos y memcached). Por tanto se han cumplido los objetivos planteados: la clasificación automática de imágenes médicas, la construcción modular de la aplicación y la portabilidad de la misma.

Además se han cumplido objetivos adicionales como obtener una folksonomía de imágenes médicas, obtenida de manera satisfactoria gracias al desarrollo de la aplicación *FlickrBackup*, o la inclusión de memcached en el sistema.

5.2. Líneas Futuras.

Esta sección intenta indicar que posibles líneas de desarrollo se pueden seguir para continuar con la labor llevada a cabo en este proyecto.

Se han pensado las siguientes líneas futuras.

- El desarrollo de diferentes implementaciones para los módulos que componen *ACoAR*. Por ejemplo, diferentes métodos de *clustering*, de medidas de similaridad, etc.
- Continuar con la implementación de más métodos de *ACoAR* desarrollados por Francisco Echarte. Como por ejemplo la actualización de *ModelACoAR*.
- Desarrollar una interfaz gráfica a esta implementación de *ACoAR* que permita la configuración de todos los parámetros, el intercambio de módulos y hacer una monitorización de la ejecución.
- Construir esta implementación de *ACoAR* como un sistema distribuido para compartir la carga de trabajo de ciertos cálculos entre varias máquinas, como, por ejemplo, el *clustering*, el cálculo de similaridades, etc. Francisco Echarte ya hace uso de esto mediante la herramienta *Gearman*[1].
- Redefinir de alguna forma el algoritmo para, en vez de obtener una serie de conceptos planos, puedan generarse una serie de conceptos ordenados de una forma jerárquica.
- Añadir algún método que intente clasificar aquellos recursos convergidos que se han quedado sin clasificar, antes de calcular las similaridades.
- Hacer uso de Snomed-CT como ayuda a la clasificación de las imágenes, aislando términos médicos o usándolo como apoyo para jerarquizar los conceptos.

[1] http://gearman.org/

6. Bibliografía.

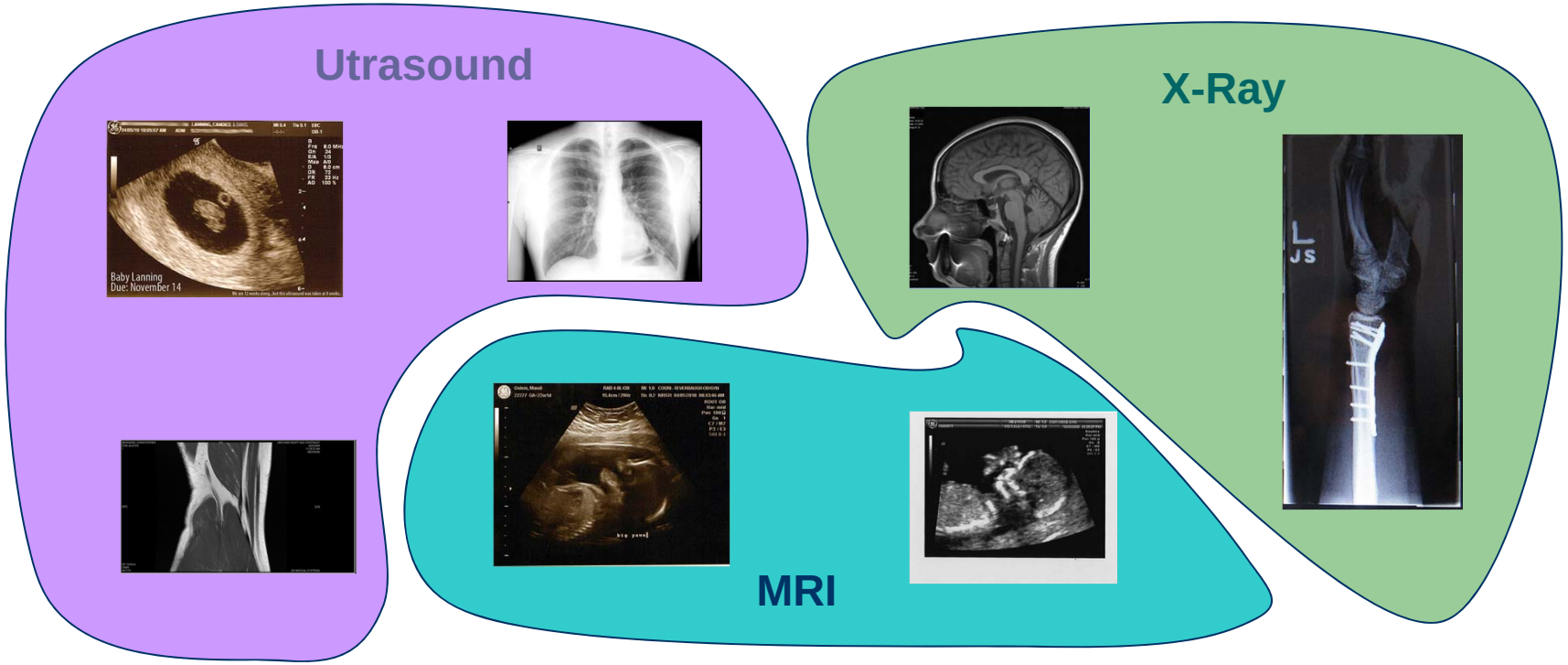
[1] Gruber, T. (2008). Collective Knowledge Systems: Where the Social Web meets the Semantic Web.
[2] Abbasi, R., Staab, S., Cimiano, P. (2007). Organizing Resources on Tagging Systems using T-ORG.
[3] Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web.
[4] Echarte, F. (2011) Un método Incremental para la Clasificación Automática y Semántica de Recursos en Sistemas de Etiquetado Colaborativo. (Departamento de Ingeniería Matemática e Informática, Universidad Pública de Navarra).
[5] Chen, M., Liu, X., Quin, J. (2008). Semantic Relation Extraction from Socially-Generated Tags: A Metodology for Metadata Generation.
[6] Angeletou, S., Sabou, M., Motta, E. (2009). Folksonomy Enrichment and Search.
[7] Bindelli, S., Criscione, C., Curino, C. A., Drago, M. L., Eynard, D., Orsi, G. (2008). Improving Search and Navigation by Combining Othologies and Social Tags.
[8] Schmitz, P. (2006). Inducing Ontology from Flickr Tags.

Sistema Colaborativo de Clasificación de Imágenes Médicas basado en Folksonomías

Adrián Ciordia Galar

Objetivos

- Aplicación:



Objetivos

- Aplicación:
  - Origen de Datos:
    - Folksonomía de Imágenes Médicas.
      - (Recursos, Etiquetas, Anotaciones)
  - Haciendo uso de:
    - Tesis doctoral de Francisco Echarte (*ACoAR*).

Objetivos

- Objetivos Adicionales:
  - Aplicación Modular.
  - Fácil Portabilidad.

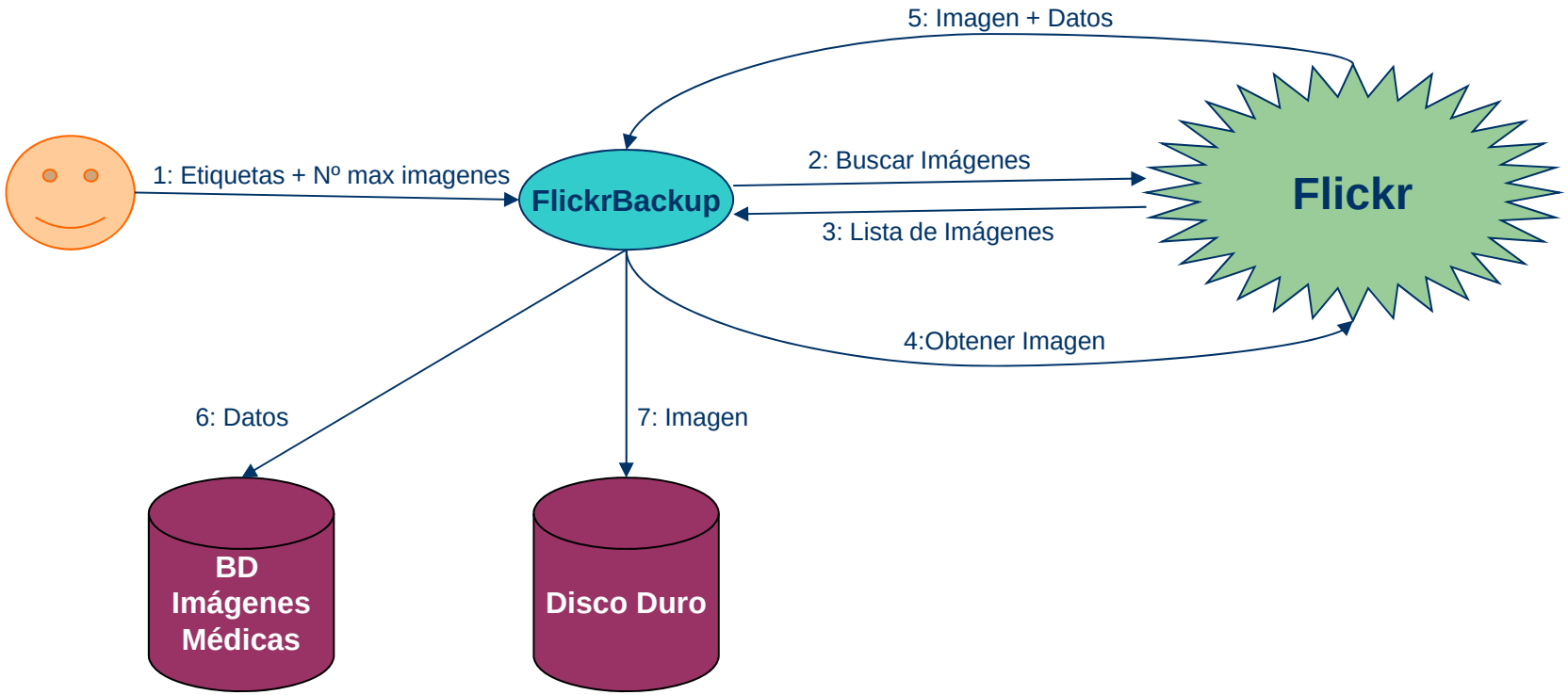
Obtención de la Folksonomía

- Primer problema:
  - Obtener la Folksonomía de Imágenes Médicas.

- Solución:
  - *FlickrBackup*.
    - Descarga imágenes de *Flickr*.

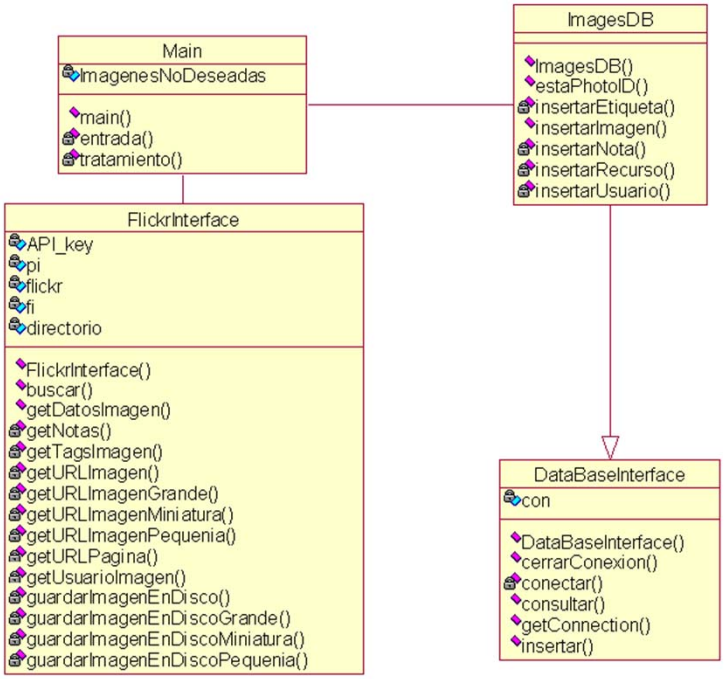
Obtención de la Folksonomía

- *FlickrBackup*:



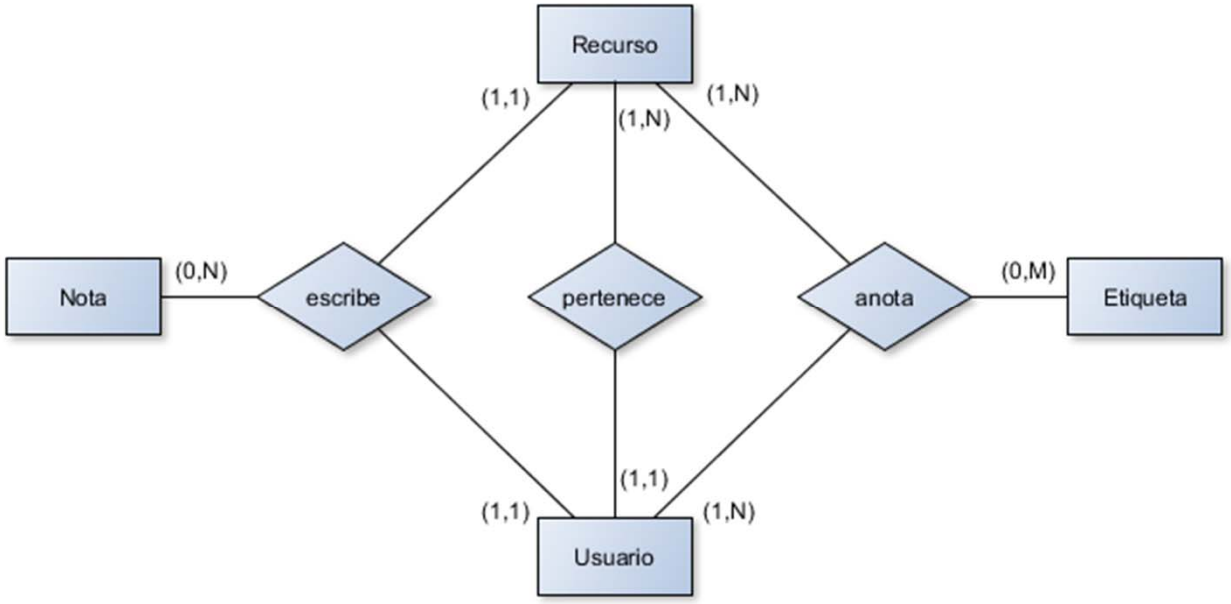
Obtención de la Folksonomía

- Diagrama de clases de *FlickrBackup*:



Obtención de la Folksonomía

- Diagrama E-R de la BD de Imágenes Médicas:



Obtención de la Folksonomía

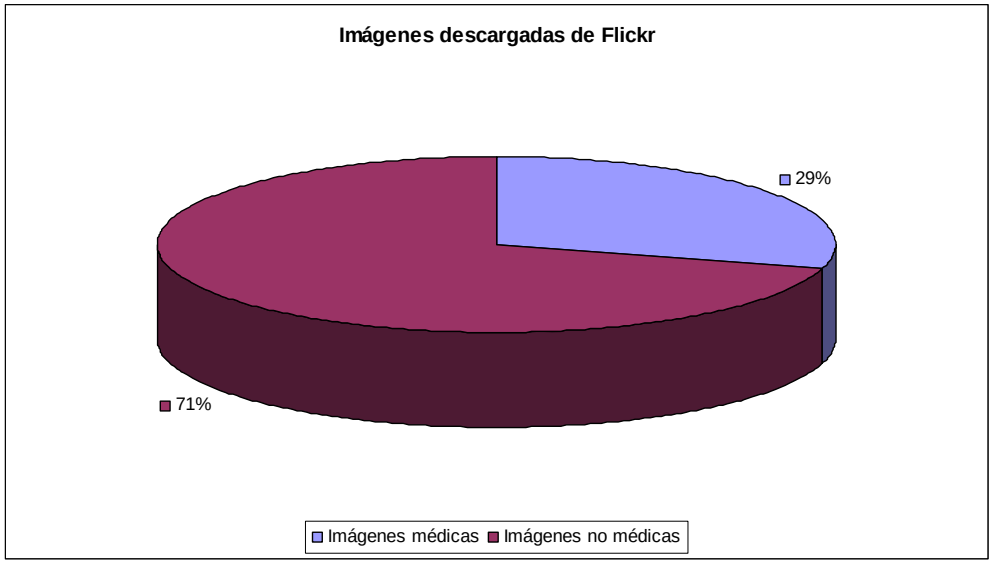
- Tecnología:
  - *Java*
  - *MySQL*
  - *flickrj* (API de *Flickr* para Java)

Obtención de la Folksonomía

- Obtención de imágenes:
  1. Introducir conjunto de etiquetas.
  2. Descargar imágenes.
  3. Eliminar imágenes no deseadas.
  4. Vuelta a 1 con otro conjunto de etiquetas.

Obtención de la Folksonomía

- Resultados:
  - Imágenes totales: 29.163
  - Imágenes válidas: 8.579
- Folksonomía:
  - 8.579 Imágenes médicas
  - 5.478 Etiquetas.
  - 58.878 Anotaciones.
  - 1.988 Usuarios
  - 1.270 Notas



ACoAR

- Aplicación:
  - Basada en la tesis de Francisco Echarte.
  - Adaptada a:
    - Imágenes médicas.
    - La tecnología usada para su implementación.
    - Un solo proceso.
    - Entorno de ejecución.

ACoAR

- Entrada:
  - Recursos (R).
  - Etiquetas (T).
  - Anotaciones.

- Salida:
  - Conceptos (C).
  - Clasificación (Z).
  - Similaridades:
    - Concepto-Concepto (SCC).
    - Concepto-Recurso (SCR).
    - Concepto-Diccionario (SCD).
    - Recurso-Recurso (SRR).

- Elementos Internos:
  - Diccionario (D).
  - Vectores Representativos:
    - de Recursos (VR).
    - de Concepto (VC).
    - de Diccionario (VD).

ACoAR

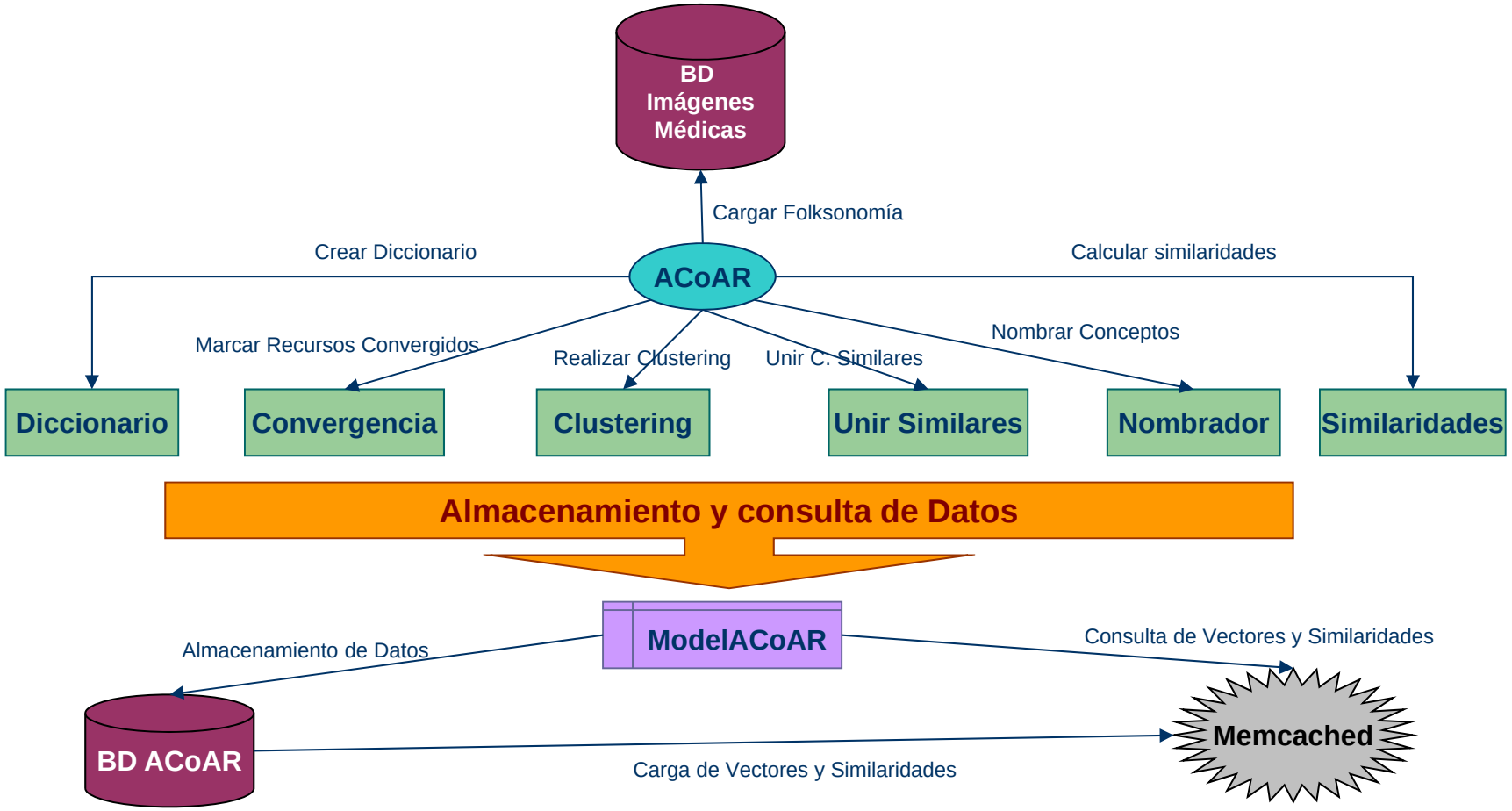
- Diccionario (D):
  - Subconjunto de T:
    - Etiquetas deben tener un mínimo de anotaciones.
  - Define un espacio de |D| dimensiones.
    - Se representan recursos, conceptos y etiquetas.
- Vectores:

Etiqueta del Diccionario

D1	D2	D3	D4	D5	D6	D7	D8	D9
100	0	25	1520	70	0	0	1	2450

Nº de Anotaciones

ACoAR



ACoAR

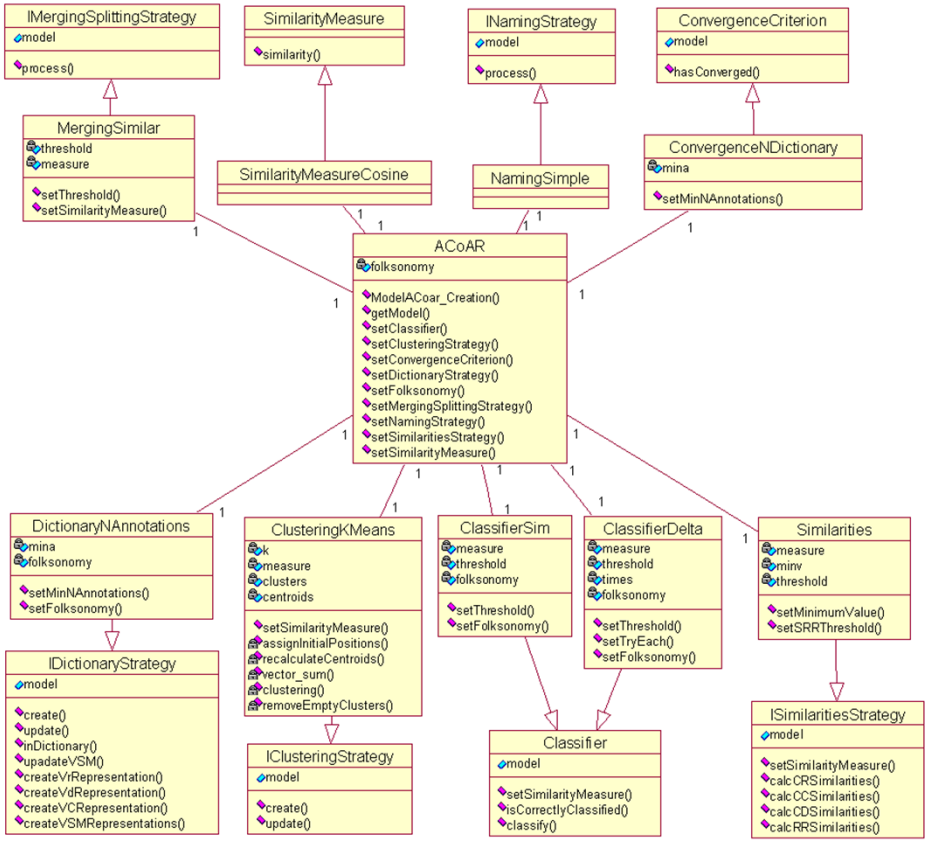
- Módulos:
  - Diccionario: ***DictionaryNAnnotations***
    - Anotaciones etiquetas >= umbral
    - Calcula *VR*, *VD* y *VC*.
  - Convergencia: ***ConvergenceNDictionary***
    - Suma componentes VR >= umbral
  - *Clustering*: ***ClusteringKMeans***
    - *KMeans*
    - *Crea C y Z.*
  - Unir Similares: ***MergingSimilar***
    - Une si SCC >= umbral
  - Nombrador: ***NamingSimple***
    - Concatena Etiquetas más representativas
  - Similaridades: ***Similarities***
    - Calcula similaridad entre parejas de elementos

ACoAR

- Módulos Secundarios:
  - Clasificador: ***ClassifierSim***
    - Clasifica si similaridad concepto-recurso >= umbral
  - Clasificador: ***ClassifierDelta***
    - Clasifica si *delta* >= umbral
    - *Delta* = diferencia similaridades entre los 2 mejores conceptos candidatos
  - Medida de Similaridad: ***SimilarityMeasureCosine***
    - Medida de similaridad del coseno

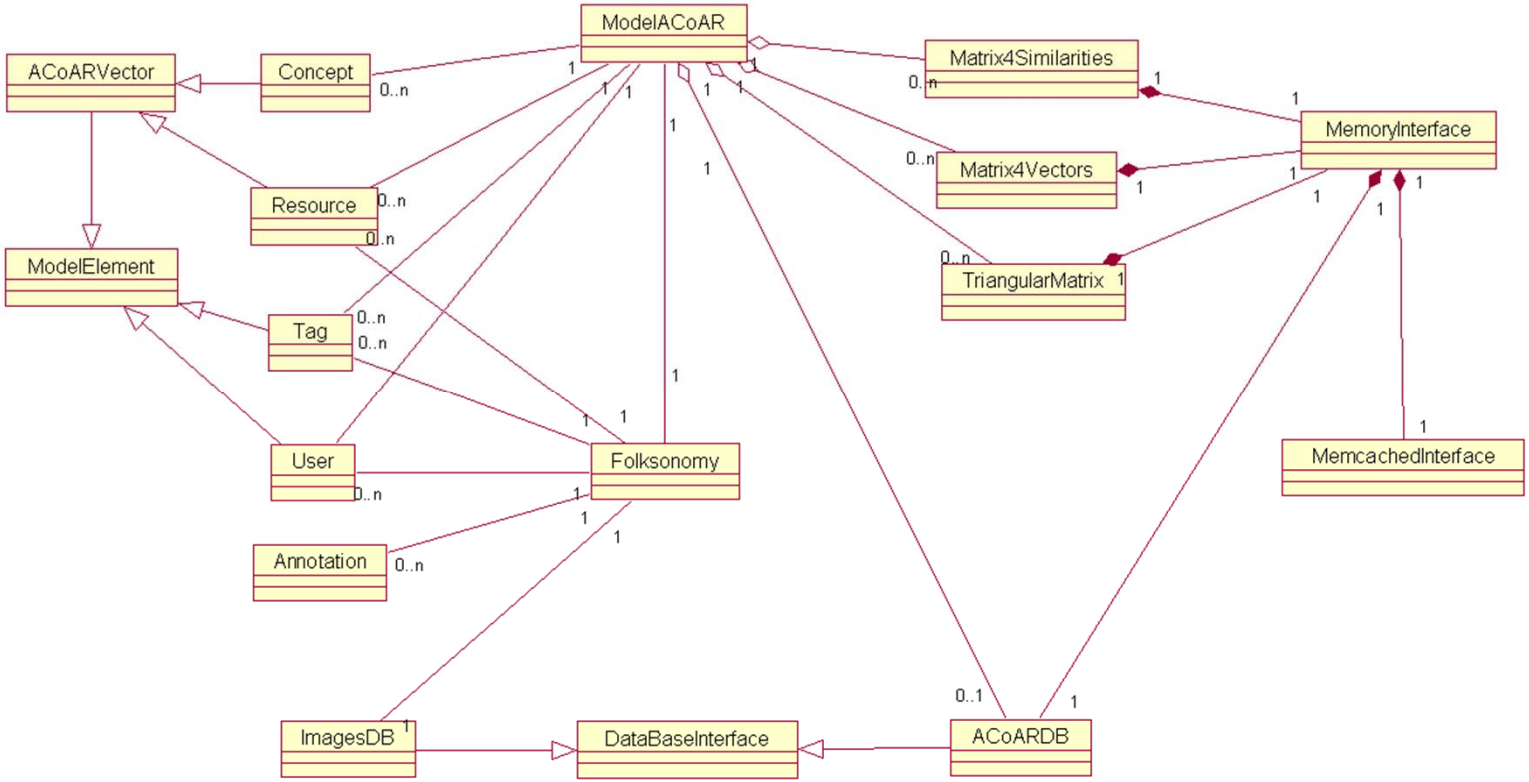
ACoAR

- Estructura Principal:



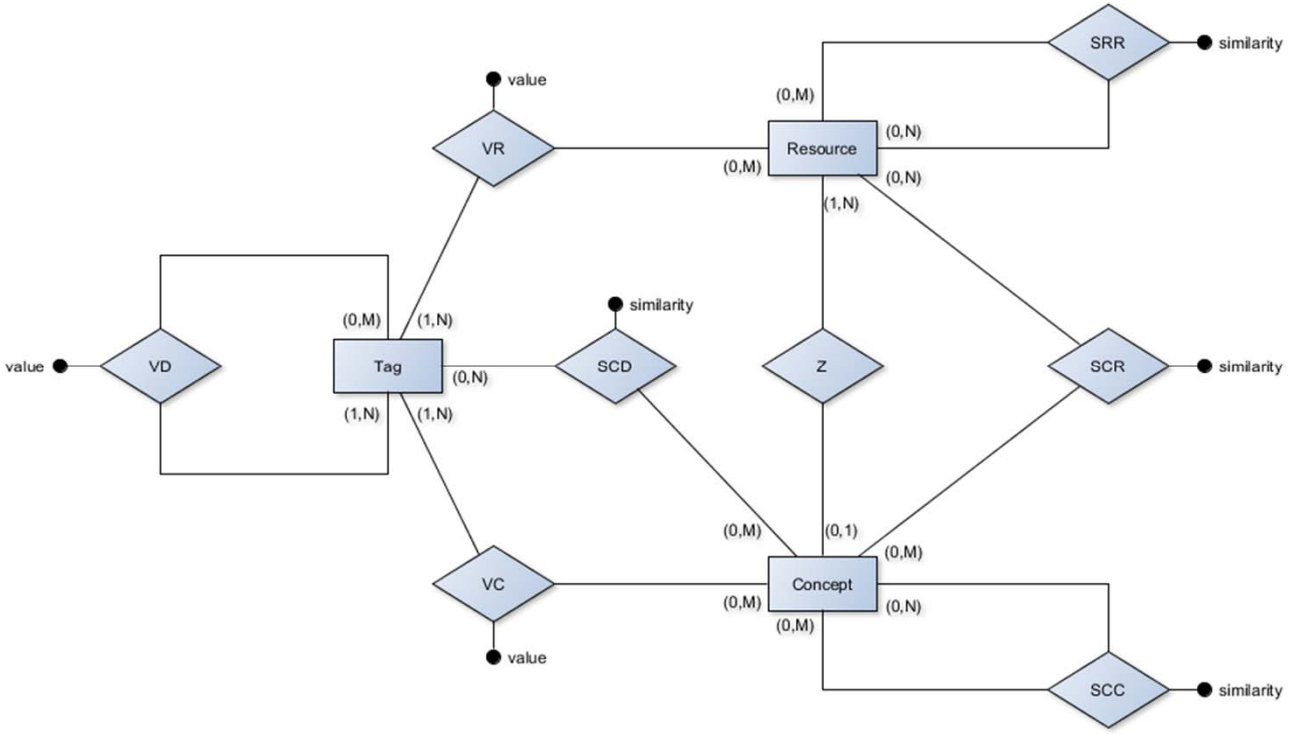
ACoAR

- Modelo de Datos:



ACoAR

- Diagrama E-R BD ACoAR:

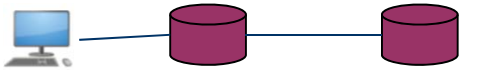


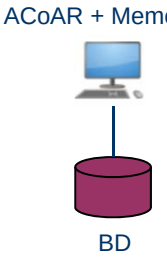
ACoAR

- Tecnología:
  - *Java*
  - *MySQL*
  - *Membase* (*Memcached*)
  - *spymemcached* (API de *Memcached*)

ACoAR

- Problemas:
  - Picos de Memoria:
    - Causa: Similaridades recurso-recurso (SRR)
    - Solución: *Memcached*
      - Varias pruebas:





  - Solución óptima:
    - *Memcached* en la misma máquina donde se ejecuta *ACoAR* como capa intermedia en la base de datos.
    - Cálculos más lentos → espera respuesta *Memcached*.

ACoAR

- Problemas:
  - Tiempos de ejecución:
    - Causa: cálculo de similaridades recurso-recurso (SRR)
    - Solución: no calcular todas las combinaciones de recursos
      - SRR:
        - para recursos de un mismo concepto
        - Recursos de conceptos con SCC >= umbral.

ACoAR

- Otros cambios de diseño:
  - Criterio de Convergencia:
    - Original: Nº de anotaciones de un recurso >= umbral
    - Nuevo: Nº de anotaciones de D en el recurso >= umbral
      - Más rápido
      - Sólo hay que sumar las componentes de VR

Resultados Experimentales

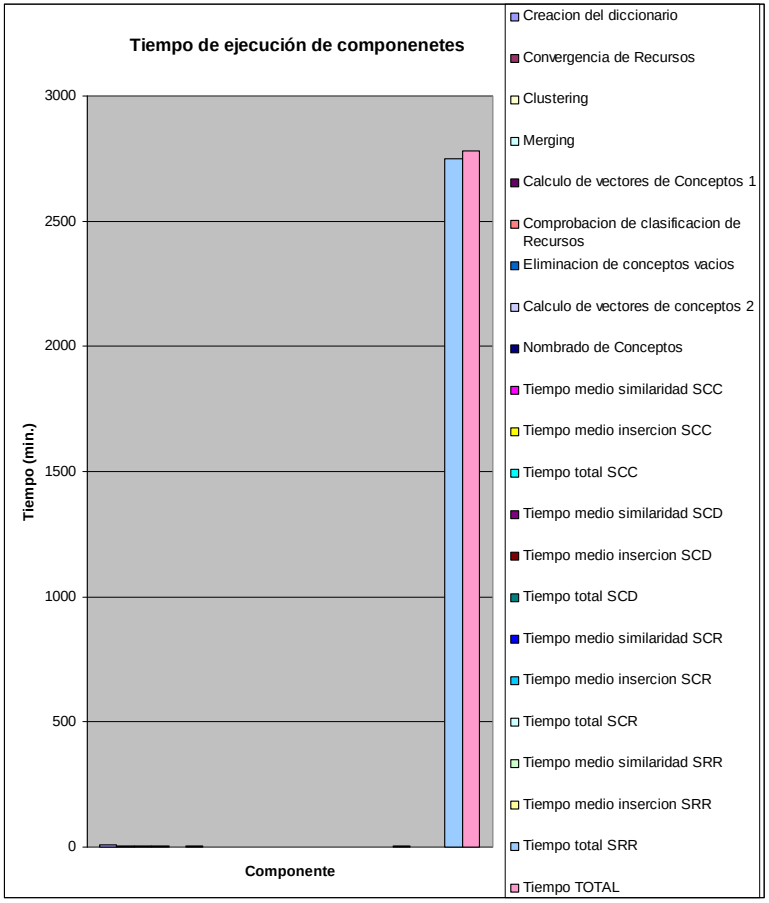
- Parámetros:
  - ***dictionaryMina***: umbral para que una etiqueta pertenezca a D.
  - ***convergenceMina***: umbral para que converja un recurso.
  - ***mergingThreshold***: umbral para unir 2 conceptos.
  - ***classifierThreshold***: umbral para clasificar un recurso en un concepto.
  - ***similaritiesMinv***: valor mínimo de similaridad para se almacenada.
  - ***similaitiesSRRThres***: valor mínimo de *SCC* para calcular SRR.
  - ***classifierT***: tipo de clasificador. *Sim* o *Delta*.

Resultados Experimentales

- 25 Pruebas
- Combinando valores de los parámetros:
  - *dictionaryMina*
  - *mergingThreshold*
  - *classifierThreshold*
  - *similaritiesMinv*
  - *classifierT*
- Valores fijos
  - *convergenceMina* = 1
  - *similaritiesSRRThres* = 0,4

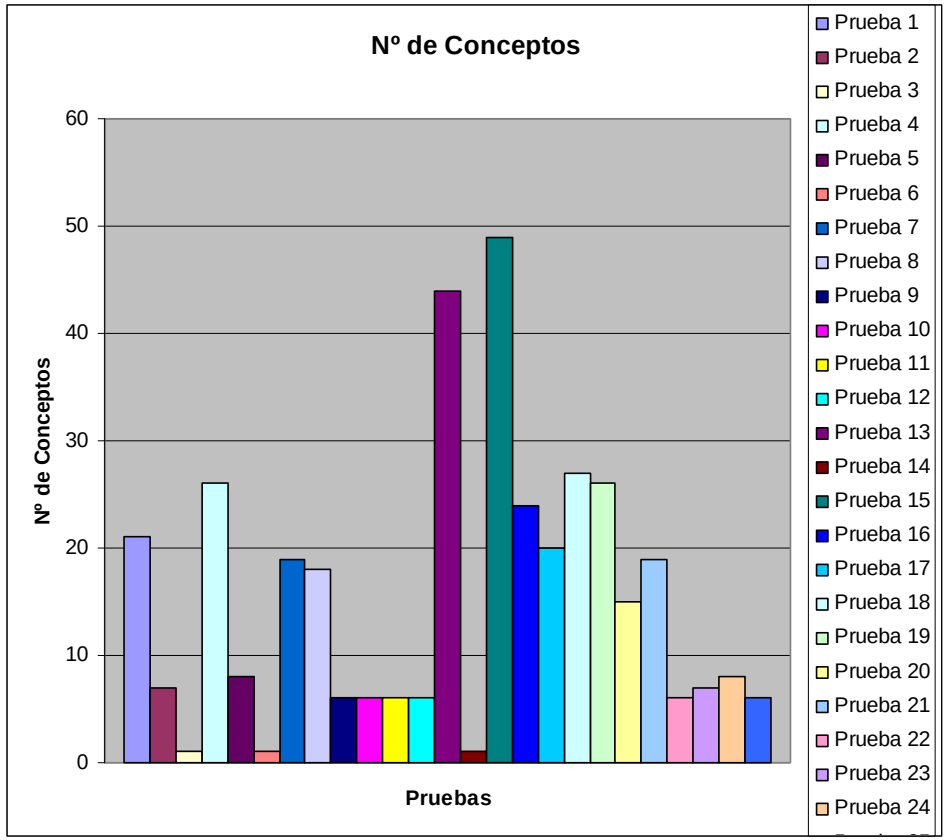
Resultados Experimentales

- Tiempos de Ejecución:
  - Mayoría de Tiempo empleado para SRR



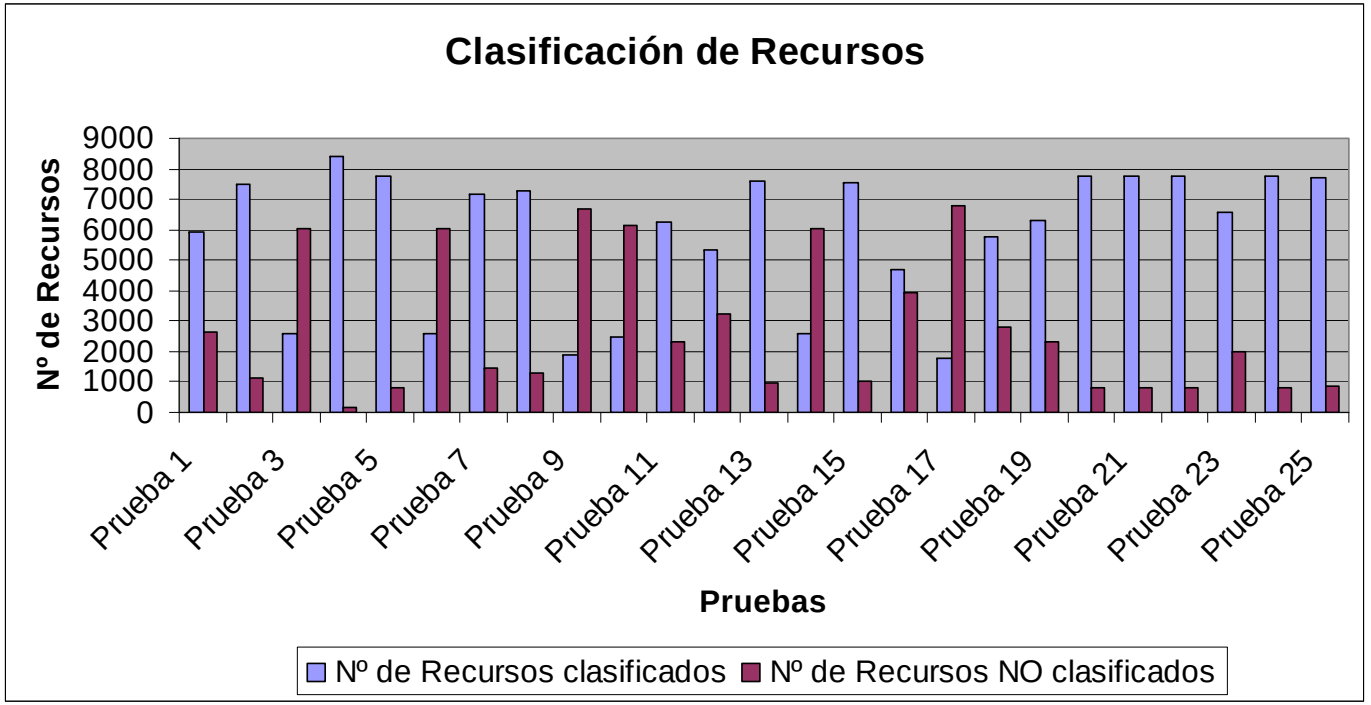
Resultados Experimentales

- Nº de Conceptos:
  - En relación con:
    - Tamaño de D
    - *mergingThreshold*



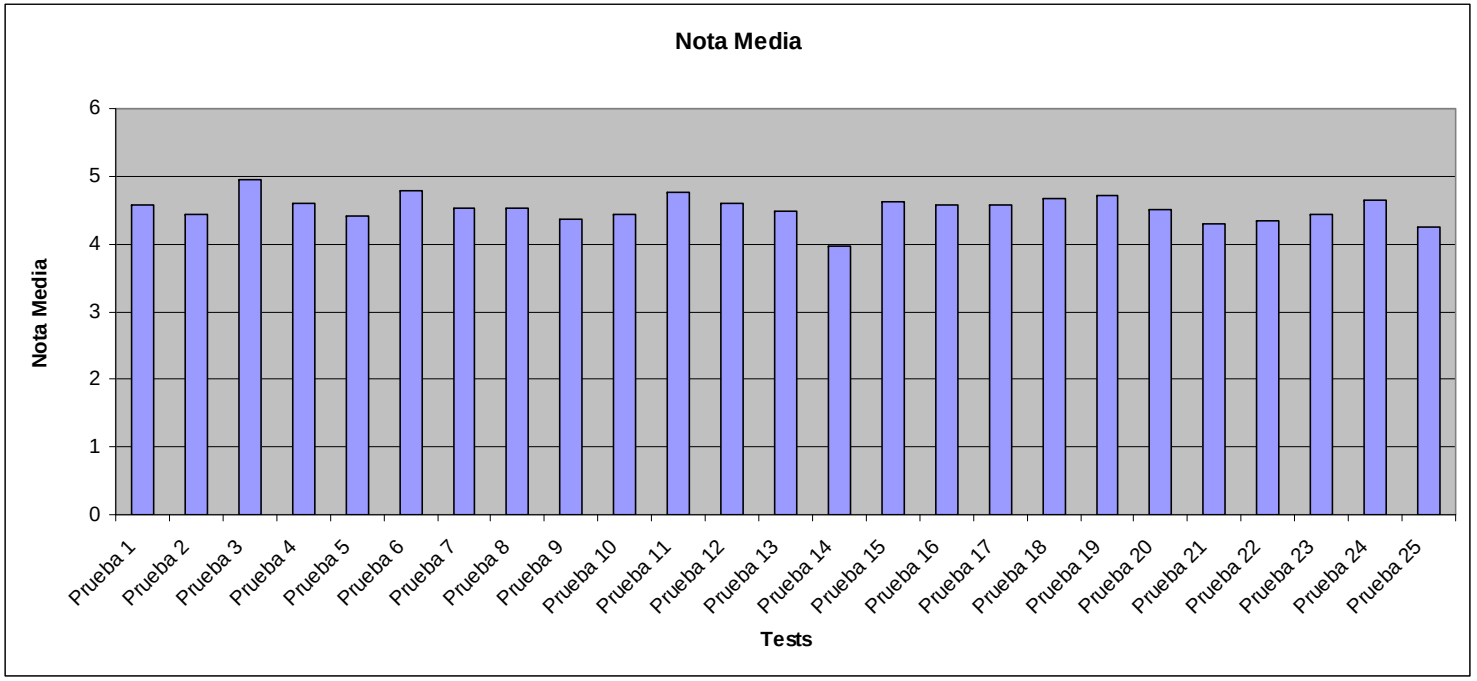
Resultados Experimentales

- Clasificación de Recursos:
  - Baja cuando: *dictionaryMina* bajo o *mergingThreshold* alto



Resultados Experimentales

- Clasificación de Recursos:
  - Nota entre 1 (lo peor) y 5 (lo mejor).



Conclusiones

- Tamaño del diccionario influye (*dictionaryMina*):
  - Tiempo:
    - de creación del diccionario.
    - de *clustering*.
    - de cálculo de vectores.
    - de cálculo de similaridades.
  - Nº de conceptos
    - También condicionado por *mergingThreshold*.
    - Si D es muy pequeño, no etiquetas para crear ni conceptos muy genéricos.

Conclusiones

- Clasificador *Sim*: mayor porcentaje de recursos clasificados.
  - Con *Delta*, *classifierThreshold* más sensible.
- Recursos bien clasificados en todas las pruebas.

Tiempo	Causa
Clasificación Recursos	Nº Recursos
SCR	Nº de Recursos
SRR	Nº de Recursos, Conceptos y SCC
Nombrado de Conceptos	Nº de Conceptos
SCC	Nº de Conceptos
SCD	Tamaño D y Nº de Conceptos

Conclusiones

- Problemas Tiempo:
  - Cálculo de SRR.
    - Calcularlo sólo cuando sea necesario.
- Problemas Conceptos:
  - Asocia etiquetas que no tienen relación
    - Ej: *brain* y *knee*.
  - Uso de etiquetas no semánticas:
    - Ej: *or* y *of*
  - Estos problemas normalmente asociados a un diccionario grande.

Conclusiones

- Resumen:
  - Se ha desarrollado una aplicación:
    - Clasifica imágenes médicas en conceptos.
    - Calcula similaridades.
    - Diseño modular.
    - Fácil Portabilidad.
  - Elementos más influyentes:
    - Tamaño del Diccionario.
    - Nº de Recursos.
  - Problemas:
    - Tiempo de ejecución SRR.
    - Conceptos:
      - Mezcla etiquetas poco relacionadas.
      - Uso de Etiquetas no semánticas.